

PROCEEDINGS
SEKE 2015

**The 27th International Conference on
Software Engineering &
Knowledge Engineering**

Sponsored by

KSI Research Inc. and Knowledge Systems Institute Graduate School, USA

Technical Program

July 6 – 8, 2015

Wyndham Pittsburgh University Center, Pittsburgh, USA

Organized by

KSI Research Inc. and Knowledge Systems Institute Graduate School, USA

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher.

ISBN: 1-891706-37-3

ISSN: 2325-9000 (print)

2325-9086 (online)

Additional copies can be ordered from:

KSI Research Inc. and Knowledge Systems Institute Graduate School

3420 Main Street

Skokie, IL 60076 USA

Tel: +1-847-679-3135

Fax: +1-847-679-3166

Email: seke@ksiresearch.org

Web: <http://www.ksi.edu>

Proceedings preparation, editing and printing are sponsored by KSI Research Inc. and Knowledge Systems Institute Graduate School, USA.

Printed by KSI Research Inc. and Knowledge Systems Institute Graduate School

FOREWORD

Welcome to the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE), in Pittsburgh, Pennsylvania, USA, the City of Champions. In over a quarter of century, SEKE has established itself as a major international forum to foster, among academia, industry, and government agencies, discussion and exchange of ideas, research results and experience in software engineering and knowledge engineering. The SEKE community has grown to become a very important and influential source of ideas and innovations on the interplays between software engineering and knowledge engineering, and its impact on the knowledge economy has been felt worldwide. On behalf of the Program Committee Co-Chairs and the entire Program Committee, it is my great pleasure to invite you to participate, not only in the technical program of SEKE 2015 and its rich assortment of activities, but also in enjoying the beautiful and historical Steel City and people of Pittsburgh.

This year, we received 238 submissions from 35 countries. Through a rigorous review process where a majority (90 percent) of the submitted papers received three reviews, and the rest with two reviews, we were able to select 69 full papers for the general conference (29 percent), and 80 short papers (34 percent). Out of that, 6 papers have been accepted for special tracks, and 128 papers are scheduled for presentation in thirty-nine sessions during the conference. In addition, the technical program includes excellent keynote speeches, poster and demo presentations, as well as special tracks: Software Assurance, Intelligent Transportation Systems, and Testing.

The high quality of the SEKE 2015 technical program would not have been possible without the tireless effort and hard work of many individuals. First of all, I would like to express my sincere appreciation to all the authors whose technical contributions have made the final technical program possible. I am very grateful to all the Program Committee members whose expertise and dedication made my responsibility that much easier. My gratitude also goes to the keynote speakers who graciously agreed to share their insight on important research issues, to the conference organizing committee members for their superb work, and to the external reviewers for their contribution.

Personally, I owe a debt of gratitude to a number of people whose help and support with the technical program and the conference organization are unfailing and indispensable. I am deeply indebted to Dr. S. K. Chang, Chair of the Steering Committee, for his constant guidance and support that are essential to pull off SEKE 2015. My heartfelt appreciation goes to Dr. Marek Reformat of University of Alberta, Canada, the Conference Chair, for his help and experience, and to the Program Committee Co-Chairs, Dr. Kehan Gao of Eastern Connecticut State University, USA, and Dr. Shihong Huang of Florida Atlantic University, USA, for their outstanding team work.

I am truly grateful to the special track organizers, Dr. Dianxiang Xu of Boise State University, USA, Dr. Behrouz Far of University of Calgary, Canada, Dr. Jerry Gao of San Jose State University, USA, and Dr. Genny Tortora of University of Salerno, Italy, for their excellent job in organizing the special tracks.

I would like to express my great appreciation to all the Publicity Co-Chairs, Dr. Xiaoying Bai of Tsinghua University, China, and Dr. Jun Suzuki of University of Massachusetts Boston, USA, for their important contributions, to the Asia, Europe, India, and South America liaisons, Dr. Hironori Washizaki of Waseda University, Japan, Dr. Raul Garcia Castro of Universidad Politecnica de Madrid, Spain, Dr. Swapan Bhattacharya of National Institute of Technology Karnataka, India, and Dr. Jose Carlos Maldonado of University of Sao Paulo, Brazil, for their great efforts in helping expand the SEKE community, and to the Demo&Poster session Co-Chairs, Dr. Farshad Samimi of Enphase Energy, USA, and Dr. Dragutin Petkovic of San Francisco State University, USA, for their work.

Last but certainly not the least, I must acknowledge the important contributions that the KSI staff members have made. Their timely and dependable support and assistance throughout the entire process have been truly remarkable. It has been a great pleasure to work with all of them. Finally, I sincerely thank you for finding your way to Pittsburgh to participate in SEKE 2015, and hope you will enjoy this experience that may leave you long lasting memories.

Haiping Xu

SEKE 2015 Program Chair

SEKE 2015

The 27th International Conference on Software Engineering & Knowledge Engineering

July 6 – 8, 2015

Wyndham Pittsburgh University Center, Pittsburgh, USA

Conference Organization

CONFERENCE CHAIR

Marek Reformat, University of Alberta, Canada

PROGRAM COMMITTEE CHAIR

Haiping Xu, University of Massachusetts Dartmouth, USA

PROGRAM COMMITTEE CO-CHAIRS

Kehan Gao, Eastern Connecticut State University, USA
Shihong Huang, Florida Atlantic University, USA

STEERING COMMITTEE CHAIR

Shi-Kuo Chang, University of Pittsburgh, USA

STEERING COMMITTEE

Vic Basili, University of Maryland, USA
Bruce Buchanan, University of Pittsburgh, USA
C. V. Ramamoorthy, University of California, Berkeley, USA

ADVISORY COMMITTEE

Jerry Gao, San Jose State University, USA
Swapna Gokhale, University of Connecticut, USA
Natalia Juristo, Universidad Politecnica de Madrid, Spain
Taghi Khoshgoftaar, Florida Atlantic University, USA
Guenther Ruhe, University of Calgary, Canada
Masoud Sadjadi, Florida International University, USA
Du Zhang, California State University, USA

PROGRAM COMMITTEE

Silvia Teresita Acuna, Universidad Autonoma de Madrid, Spain
Shadi Alawneh, C-CORE, Canada
Izzat Alsmadi, Boise State University, USA
Taiseera Albalushi, Sultan Qaboos University, Oman
Mark Allison, University of Michigan - Flint, USA
John Anvik, Central Washington University, USA
Omar El Ariss, Penn State University at Harrisburg, USA
Doo-hwan Bae, Korea Advanced Institute of Science and Technology, Korea
Ebrahim Bagheri, Ryerson University, Canada
Hamid Bagheri, George Mason University and Massachusetts Institute of Technology, USA
Xiaoying Bai, Tsinghua University, China
Purushotham Bangalore, University of Alabama at Birmingham, USA
Fevzi Belli, University of Paderborn, Germany
Ateet Bhalla, Oriental Institute of Science & Technology, Bhopal, India
Swapan Bhattacharya, NITK, Surathakl, India
Alessandro Bianchi, University of Bari, Italy
Ivo Bukovsky, Czech Technical University in Prague, Czech Republic
Raul Garcia Castro, Universidad Politecnica de Madrid, Spain
Chih-Hung Chang, Hsiuping University of Science and Technology, Taiwan
Keith Chan, Hong Kong Polytechnic University, Hong Kong
Kuang-nan Chang, Eastern Kentucky University, USA
Meiru Che, University of Texas at Austin, USA
Shu-Ching Chen, Florida International University, USA
Wen-Hui Chen, National Taipei University of Technology, Taiwan
Stelvio Cimato, University of Milan, Italy
Nelly Condori-fernandez, University of Twente, The Netherlands
Fabio M. Costa, Universidade Federal de Goias, Brazil
Maria Francesca Costabile, University of Bari, Italy
Jose Luis De La Vara, Simula Research Laboratory, Norway
Massimiliano Di Penta, University of Sannio, Italy
Scott Dick, University of Alberta, Canada
Junhua Ding, East Carolina University, USA
Fei Dong, Google Inc., USA
Derek Doran, Wright State University, USA
Weichang Du, University of New Brunswick, Canada
Philippe Dugerdil, HEG - Univ. of Applied Sciences, Switzerland
Christof Ebert, Vector Consulting Services, Germany
Ali Ebneenasir, Michigan Technological University, USA
Raimund Ege, NIU, USA
Magdalini Eirinaki, San Jose State University, USA
Mahmoud Elish, King Fahd University of Petroleum and Minerals, Saudi Arabia

Davide Falessi, Cal Poly, USA
 Behrouz Far, University of Calgary, Canada
 Liana Fong, IBM, USA
 Ellen Francine Barbosa, University of Sao Paulo, Brazil
 Fulvio Frati, University of Milan, Italy
 Felix Garcia, University of Castilla-La Mancha, Spain
 Ignacio Garcia Rodriguez De Guzman, University of Castilla-La Mancha, Spain
 Swapna Gokhale, Univ. of Connecticut, USA
 Wolfgang Golubski, Zwickau University of Applied Sciences, Germany
 Desmond Greer, Queen's University Belfast, United Kingdom
 Christiane Gresse Von Wangenheim, UFSC - Federal University of Santa Catarina, Brazil
 Katarina Grolinger, University of Western Ontario, Canada
 Hassan Haghghi, Shahid Beheshti University, Iran
 Hao Han, National Institute of Informatics, Japan
 Xudong He, Florida International University, USA
 Miguel Herranz, University of Alcala, Spain
 Rubing Huang, Jiangsu University, China
 Shihong Huang, Florida Atlantic University, USA
 Bassey Isong, North-West University, South Africa
 Clinton Jeffery, University of Idaho, USA
 Jason Jung, Chung-Ang University, Korea
 Selim Kalayci, Florida International University, USA
 Marcos Kalinowski, Fluminense Federal University, Brazil
 Ananya Kanjilal, B.P. Poddar Institute of Technology and Management, India
 Eric Kasten, Michigan State University, USA
 Taghi Khoshgoftaar, Florida Atlantic University, USA
 Claudiu V. Kifor, Lucian Blaga University of Sibiu, Romania
 Jun Kong, North Dakota State University, USA
 Aneesh Krishna, Curtin University of Technology, Australia
 Vinay Kulkarni, Tata Consultancy Services, India
 Jeff Lei, University of Texas at Arlington, USA
 Meira Levy, Shenkar College of Engineering and Design, Israel
 Bixin Li, Southeast University, China
 Ming Li, Nanjing University, China
 Xin Li, Google Inc., USA
 Yuan-Fang Li, Monash University, Australia
 Zhi Li, Guangxi Normal University, China
 Jianhua Lin, Eastern Connecticut State University, USA
 Xiaoqing Frank Liu, Missouri University of Science and Technology, USA
 Shih-hsi Liu, California State University, Fresno, USA
 Ting Liu, Xian Jiaotong University, China
 Xiaodong Liu, Edinburgh Napier University, United Kingdom
 Yi Liu, Georgia College & State University, USA
 Luanna Lopes Lobato, Federal University of Goias, Brazil
 Baojun Ma, Beijing University of Posts and Telecommunications, China
 Ivan do Carmo Machado, Federal University of Bahia, Brazil
 Marcelo de Almeida Maia, Federal University of Uberlândia, Brazil
 Beatriz Marin, Universidad Diego Portales, Chile
 Riccardo Martoglia, University of Modena and Reggio Emilia, Italy
 Santiago Matalonga, Universidad ORT Uruguay, Uruguay
 Hong Mei, Peking University, China
 Hsing Mei, Fu Jen Catholic University, Taiwan
 Andre Menolli, Universidade Estadual do Norte do Parana (UENP), Brazil
 Ali Mili, NJIT, USA
 Alok Mishra, Atılım University, Turkey
 Manuel Mora, Autonomous University of Aguascalientes, Mexico

Kia Ng, ICSRiM - University of Leeds, United Kingdom
 Allen Nikora, Jet Propulsion Laboratory, USA
 Amjad Nusayr, University of Houston-Victoria, USA
 Edson Oliveira Jr, State University of Maringá, Brazil
 Xin Peng, Fudan University, China
 Oscar Pereira, University of Aveiro, Portugal
 Antonio Piccinno, University of Bari, Italy
 Alfonso Pierantonio, University of L'Aquila, Italy
 Daniel Plante, Stetson University, USA
 Rick Rabiser, Johannes Kepler University, Austria
 Filip Radulovic, Universidad Politécnica de Madrid, Spain
 Damith C. Rajapakse, National University of Singapore, Singapore
 Rajeev Raje, IUPUI, USA
 Henrique Rebelo, Universidade Federal de Pernambuco, Brazil
 Marek Reformat, University of Alberta, Canada
 Robert G. Reynolds, Wayne State University, USA
 Elder Macedo Rodrigues, Pontifical Catholic University of Rio Grande do Sul, Brazil
 Daniel Rodriguez, Universidad de Alcala, Spain
 Ivan Rodero, The State University of New Jersey, USA
 Samira Sadaoui, University of Regina, Canada
 Masoud Sadjadi, Florida International University, USA
 Claudio Sant'Anna, Universidade Federal da Bahia, Brazil
 Akila Sarirete, Effat University, Saudi Arabia
 Abdelhak-Djamel Seriai, University of Montpellier 2 for Sciences and Technology, France
 Andreas Schoenberger, Siemens AG, Germany
 Michael Shin, Texas Tech University, USA
 Martin Solari, Universidad ORT Uruguay, Uruguay
 Qinbao Song, Xi'an Jiaotong University, China
 George Spanoudakis, City University, United Kingdom
 Jing Sun, University of Auckland, New Zealand
 Yanchun Sun, Peking University, China
 Gerson Sunye, University of Nantes, France
 Jeff Tian, Southern Methodist University, USA
 Genny Tortora, University of Salerno, Italy
 Mark Trakhtenbrot, Holon Institute of Technology, Israel
 Peter Troeger, TU Chemnitz, Germany
 T. H. Tse, The University of Hong Kong, Hong Kong
 Burak Turhan, Oulu University, Finland
 Christelle Urtado, Ecole des Mines d'Alès, France
 Sylvain Vauttier, Ecole des Mines d'Alès, France
 Silvia Vergilio, Federal University of Parana (UFPR), Brazil
 Sergiy Vilkomir, East Carolina University, USA
 Aaron Visaggio, University of Sannio, Italy
 Arndt Von Staa, PUC-Rio, Brazil
 Gurisimran Walia, North Dakota State University, USA
 Huanjing Wang, Western Kentucky University, USA
 Jiacun Wang, Monmouth University, USA
 Linzhang Wang, Nanjing University, China
 Xiaoyin Wang, University of Texas at San Antonio, USA
 Ye Wang, Zhejiang Gongshang University, China
 Yong Wang, New Mexico Highlands University, USA
 Hironori Washizaki, Waseda University, Japan
 Victor Winter, University of Nebraska at Omaha, USA
 Guido Wirtz, Bamberg University, Germany
 W. Eric Wong, University of Texas at Dallas, USA
 Franz Wotawa, TU Graz, Austria

Dianxiang Xu, Boise State University, USA
Frank Weifeng Xu, Gannon University, USA
Haiping Xu, University of Massachusetts Dartmouth, USA
Zhiwei Xu, University of Michigan - Dearborn, USA
Guowei Yang, Texas State University, USA
Hongji Yang, Bath Spa University, United Kingdom
Huiqun Yu, East China University of Science and Technology, China
Jiang Yue, Fujian Normal University, China
Du Zhang, California State University, USA
Yong Zhang, Tsinghua University, China
Zhenyu Zhang, Institute of Software, Chinese Academy of Sciences, China
Jiang Zheng, ABB US Corporate Research Center, USA
Jianlin Zhu, South-Central University for Nationalities, China
Xingquan Zhu, Florida Atlantic University, USA
Eugenio Zimeo, University of Sannio, Italy

DEMO&POSTER SESSIONS CO-CHAIRS

Farshad Samimi, Enphase Energy, USA
Dragutin Petkovic, San Francisco State University, USA

PUBLICITY CHAIR

Xiaoying Bai, Tsinghua University, China
Jun Suzuki, University of Massachusetts Boston, USA

ASIA LIAISON

Hironori Washizaki, Waseda University, Japan

EUROPE LIAISON

Raul Garcia Castro, Universidad Politecnica de Madrid, Spain

INDIA LIAISON

Swapan Bhattacharya, National Institute of Technology Karnataka, Surathakl, India

SOUTH AMERICA LIAISON

Jose Carlos Maldonado, ICMC-USP, Brazil

Keynote

Model Checking Hybrid Systems

Edmund M. Clarke
(Joint work with Sicun Gao and Soonho Kong)
Carnegie Mellon University
Pittsburgh, USA

Abstract: Although every undergraduate in computer science learns about Turing Machines, it is not well known that they were originally proposed as a means of characterizing computable real numbers. For a long time, formal verification paid little attention to computational applications that involve the manipulation of continuous quantities, even though such applications are ubiquitous. In recent years, however, there has been great interest in safety-critical hybrid systems involving both discrete and continuous behaviors, including autonomous automotive and aerospace applications, medical devices of various sorts, control programs for electric power plants, etc. As a result, the formal analysis of numerical computation can no longer be ignored. In this talk, we focus on one of the most successful verification techniques, bounded model checking. Current industrial model checkers do not scale to handle realistic hybrid systems. We believe that the key to handling more complex systems is to make better use of the theory of the computable reals and computable analysis. We argue that new formal methods for hybrid systems should combine existing discrete methods in model checking with new algorithms based on computable analysis. In particular, we discuss a model checker that we are currently developing along these lines.

About the Speaker

Edmund M. Clarke is the FORE Systems University Professor of Computer Science at Carnegie Mellon University. He received his Ph.D. from Cornell University and taught at Duke and Harvard Universities before joining CMU in 1982. His research interests include hardware and software verification and automatic theorem proving. In particular, his research group developed Symbolic Model Checking using BDDs, Bounded Model Checking using fast CNF satisfiability solvers, and pioneered the use of CounterExample-Guided-Abstraction-Refinement (CEGAR). He is a co-founder of the conference on Computer Aided Verification (CAV). He has received numerous awards for his contributions to formal verification of hardware and software correctness, including the IEEE Goode Award, the ACM Kanellakis Award, the ACM Turing Award, and the CADE Herbrand Award. Dr. Clarke is a member of the National Academy of Engineering and the American Academy of Arts and Sciences. Most recently he received the 2014 Franklin Institute Bower Award and Prize for Achievement in Science for verification of computer systems.

Keynote

Development of Active Safety Assurance Technologies for Rail Intelligent Transportation System in China

Yong Qin
State Key laboratory of Rail Traffic Control and Safety
Beijing Jiaotong University
Beijing, China

Abstract: China has built the largest scale of high-speed railway in the world in recent years and will continuously invest in rail infrastructure for intra-city and inter-city transportation. For improving the efficiency and safety of these large-scale of rail network operation, rail intelligent transportation system (RITS) based on the advanced information and knowledge engineering techniques is a good solution. In this talk, the definition, architecture and key techniques of RITS are introduced. Because the railway operation safety is the most important field in RITS and focus on accident preventive ability now, many new active safety assurance technologies have been studied and applied into the practice of China railway. The intelligent fault diagnosis method based on safety region and support vector machine(SVM) algorithm is introduced to the online monitoring of the train operation status. The intelligent optimization method based on fuzzy particle swarm optimization algorithm is produced to the train traffic plan adjustment. And the risk assessment method based on fuzzy-TOPSIS is introduced to the rail network real time risk analysis. The relative China railway applications will be demonstrated in this talk.

About the Speaker

Yong Qin is the Dr., Professor of State Key laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University. He also is the vice director of Beijing Research Center of Urban Traffic Information Intelligent Sensing and Service Technologies, the vice dean and secretary general of Rail Transportation Electro-technical Committee of China Electro-technical Society, the vice dean and secretary general of Rail Intelligent Transportation Systems Committee of China Intelligent Transportation Systems Society, and the member of IEEE. His research interests are in the area of intelligent transportation systems, railway operation safety and reliability, rail network management and traffic model. He has authored or coauthored more than 100 publication papers and 5 books, has 10 patents granted, also won 7 science and technology progress award of ministry.

Keynote

Open Product Innovation

Guenther Ruhe
University of Calgary
Canada

Abstract: Offering innovative products or services is the ultimate goal of any software system development. The paradigm of Open Innovation opens a new door how this can be achieved. Leveraging all the emerging opportunities of open sourcing, outsourcing, off-shoring, crowdsourcing and social media represents a significant paradigm shift in gathering information. It facilitates the usage of external knowledge and resources. However, more information does not automatically imply making things better. Continuously analyzing data is part of the whole innovation processes. Very specific and up-to-date information from different sources needs to be synthesized to create innovative products. The talk discusses and illustrates how this analytics driven processes is designed and how is applied for software product development.

About the Speaker

Guenther Ruhe is a Professor at the University of Calgary in Canada. He received a doctorate rer. nat. degree in Mathematics with emphasis on Operations Research from Freiberg University and a doctorate habil. nat. degree (Computer Science) from University of Kaiserslautern (Germany). From 1996 until 2001, he was the deputy director of the Fraunhofer Institute for Experimental Software Engineering Fh IESE. Since 2007, he serves as an Associate Editor of the Journal of Information and Software Technology. His main research interests are in the areas of Product and Project Management, Data Analytics, Open Innovation, Decision Support in Requirements Engineering and Empirical Software Engineering. Guenther is the Founder and CEO of Expert Decisions Inc., a University of Calgary spin-off company created in 2003.

Panel Discussion

SEKE vs. Big Data

Session chair and moderator

Shi-Kuo Chang, University of Pittsburgh, USA (chang@cs.pitt.edu)

Panelists

Alexandros Labrinidis, University of Pittsburgh, USA (labrinid@cs.pitt.edu)

Zewyu Gao, San Jose State University, USA (jerry.gao@sjsu.edu)

Gregory Kapfhammer, Allegheny College, USA (gkapfham@allegheny.edu)

Panel Description: Although there are many interesting new approaches and techniques in software engineering and knowledge engineering, it is as yet unclear how SEKE can fruitfully incorporate recent research advances in Big Data. There can be many different viewpoints. One can discuss emerging research, give examples of happy marriage of SEKE and Big Data, and also provide horror stories of unfruitful application of Big Data to SEKE or vice versa. The panelists will present their views. Comments from the audience are also welcome.

Position Statements from the Panelists

Alexandros Labrinidis: *The Big Data - Same Humans Problem* Big data is transforming all aspects of the human experience, be it everyday life, scientific exploration and discovery, medicine, business, law, journalism, and decision-making at all levels of government. Despite the increases in computing technology and availability/demand for data in the last few decades, the performance of one critical component in the data processing pipeline has remained roughly the same. Namely, the ability of humans to process data has not changed significantly in the last few decades. We refer to this disparity as: "the big data - same humans problem." Therefore, when talking about the scalability aspect of big data, we need to make the distinction between two different types: (a) scalability from a systems point of view – i.e., traditional scalability/performance measures such as response time, throughput, scale-up, scale-out, etc., and (b) scalability from a human's point of view – i.e., how well the system is making sure that human users do not get lost in a sea of data. We believe that scalability from a human's point of view will soon become a requirement for successful big data systems and applications, and that it will soon be used as an optimization metric for both end-users and application programmers alike.

Zewyu Gao: *Engineering Secured, Reliable and High Quality Big Data Real-Time Application Systems and Services - Emergent Issues and Needs in Quality of Services* According to IDC, the big data and analytics market will reach \$125 billion worldwide in 2015. The fast advances of Big Data Technology, Analytics Solutions, and Application Systems provide great business opportunities and strong

demands in building secure and reliable high-quality big data application systems and services. This raises many important issues and needs in system quality assurance for big data applications and services. Unfortunately, conventional software validation methods and QoS techniques are not adequate to cope with the special features of big data application and analytics systems. This panel discussion will pay attention to the major differences and features in QoS between conventional software systems and big data analytics systems. This talk will focus on these issues, challenges and needs, and provide related research topics and directions in future

Gregory Kapfhammer: *Is Big Data a Big Deal? Not Without Correct Software!* Big data analytics software allows researchers and practitioners to create descriptive models and make predictions. Often characterized by the "three Vs" of volume, velocity, and variety, big data systems must respectively handle large amounts of data that arrive rapidly and take many different forms. In fields such as evidence-based medicine and the detection of financial fraud, big data software is poised to, and indeed already is, making important contributions. However, there is an additional "V" that is often overlooked by both researchers and practitioners: veracity. That is, if there is a lack of correctness in the software and data that make up a big data analytics system, then the data models and the resulting predictions may be compromised — with serious consequences.

The challenge for software testing researchers is to develop and empirically evaluate new methods that can accommodate the volume, velocity, and variety that is characteristic of big data systems. While some preliminary work (e.g., the testing of both data mining systems and database applications) has recently been published, few software engineering researchers have focused on big data testing. Since veracity is not always considered by big data researchers, the challenge for these individuals is to create and assess new techniques that, whenever possible, holistically consider all of the "four Vs". If not already doing so, practitioners in both of these fields should start to establish a confidence in the correctness of both their software and data through the disciplined use of testing.

Table of Contents

Foreward.....	iii
Conference Organization.....	iv
KeyNote 1: Model Checking Hybrid Systems, Professor Edmund M. Clarke	ix
KeyNote 2: Development of Active Safety Assurance Technologies for Rail Intelligent Transportation System in China, Professor Yong Qin.....	x
KeyNote 3: Open Product Innovation, Professor Guenther Ruhe	xi
Panel Discussion: SEKE vs. Big Data, Professor Shi-kuo Chang, Professor Alexandros Labrinidis, Professor Zewyu Gao, Professor Gregory Kapfhammer.....	xii

Requirements Engineering I

Towards Building Knowledge on Causes of Critical Requirements Engineering Problems ..	1
<i>Marcos Kalinowski, Rodrigo Spinola, Tayana Conte, Rafael Prikladnicki, Daniel Méndez Fernández and Stefan Wagner</i>	
Identification and Classification of Requirements from App User Reviews (S)	7
<i>Hui Yang and Peng Liang</i>	
MoLVERIC: An Inspection Technique for MoLIC Diagrams (S)	13
<i>Adriana Lopes, Anna Beatriz Marques, Simone Barbosa and Tayana Conte</i>	

Self-Adaptive Software

A Middleware Framework for Leveraging Local and Global Adaptation in IT Ecosystems .	18
<i>Soojin Park and Young B. Park</i>	
A Framework Based on Learning Techniques for Decision-making in Self-adaptive Software (S)	24
<i>Frank José Affonso, Gustavo Leite, Rafael A. P. Oliveira and Elisa Yumi Nakagawa</i>	
Towards Knowledge-intensive Software Engineering Framework for Self-Adaptive Software (S)	30
<i>Hyo-Cheol Lee and Seok-Won Lee</i>	

Software Project Management I

How to Teach the Usage of Project Management Tools in Computer Courses: A Systematic Literature Review (S)	36
<i>Rafael Gonçalves and Christiane Gresse von Wangenheim</i>	
Soft Skills in Scrum Teams. A survey of the most valued to have by Product Owners and Scrum Masters (S).....	42
<i>Gerardo Matturro, Carina Fontán and Florencia Raschetti</i>	
Reflecting, adapting and learning in small software organizations: an action research approach (S)	46
<i>Suzana Sampaio, Marcelo Marinho, Alexandre Luna and Hermano Moura</i>	

Slow Intelligence System

Application of Slow Intelligence Framework for Smart Pet Care System Design	51
<i>Shi-Kuo Chang, Wen-Hui Chen, Wen-Chyi Lin and Christopher Lee Thomas</i>	
A Slow Intelligence System Test Bed Enhanced with Super-Components	57
<i>Shi-Kuo Chang, Sen-Hua Chang, Jun-Hui Chen, Xiao-Yu Ge, Nikolaos Katsipoulakis, Daniel Petrov and Anatoli Shein</i>	
An Adaptive Contextual Recommender System: a Slow Intelligence Perspective	64
<i>Francesco Colace, Luca Greco, Saverio Lemma, Marco Lombardi, Duncan Yung and Shi-Kuo Chang</i>	

Testing I

An Automated Testing Framework for Statistical Testing of GUI Applications	72
<i>Lan Lin, Jia He and Yufeng Xue</i>	
Test Model and Coverage Analysis for Location-based Mobile Services	80
<i>Tao Zhang, Jerry Gao, Oum-El-Kheir Aktouf and Tadahiro Uehara</i>	
Generating various contexts from permissions for testing Android applications (S)	87
<i>Kwangsik Song, Ah-Rim Han, Sehun Jeong and Sungdeok Cha</i>	

Recommendation Systems

Context-aware Recommendation System with Anonymous User Profile Learning	93
<i>Yan Liu, Yangyang Xu and Mei Chen</i>	
Recommendation in the Digital TV Domain: an Architecture based on Textual Description Analysis	99
<i>Felipe Barbosa Araújo Ramos, Antonio Alexandre Moura Costa, Reudismam Rolim, Gustavo Soares, Hyggo Oliveira de Almeida and Angelo Perkusich</i>	
A Collaborative Method to Reduce the Running Time and Accelerate the k-Nearest Neighbors Search (S)	105
<i>Antonio Alexandre Moura Costa, Reudismam Rolim de Sousa, Felipe Barbosa Araújo Ramos, Gustavo Araújo Soares, Hyggo Oliveira de Almeida and Angelo Perkusich</i>	

Cloud Computing: Security and Deployment

Achieving Efficient Access Control via XACML Policy in Cloud Computing	110
<i>Xin Pei, Huiqun Yu and Guisheng Fan</i>	
A Reliable and Secure Cloud Storage Schema Using Multiple Service Providers	116
<i>Haiping Xu and Deepti Bhalerao</i>	
Towards a Deployment System for Cloud Applications	122
<i>Ruici Luo, Wei Ye and Shikun Zhang</i>	

Empirical Software Engineering I

Impact of Unanticipated software evolution on development cost and quality: an empirical evaluation	128
<i>Rodrigo A. Vilar, Anderson Lima, Hyggo Almeida and Angelo Perkusich</i>	

An empirical study on the impact of Python dynamic features on change-proneness	134
<i>Beibei Wang, Lin Chen, Wanwangying Ma, Zhifei Chen and Baowen Xu</i>	
Evaluating Software Engineers' Acceptance of a Technique and Tool for Web Usability Inspection	140
<i>Luis Jorge Enrique Rivero Cabrejos, Auri Marcelo Rizzo Vincenzi, José Carlos Maldonado and Tayana Conte</i>	

Knowledge Management

AMBIT: Semantic Engine Foundations for Knowledge Management in Context-dependent Applications	146
<i>Riccardo Martoglia</i>	
Documenting Implementation Decisions with Code Annotations	152
<i>Tom-Michael Hesse, Arthur Kuehlwein, Barbara Paech, Tobias Roehm and Bernd Bruegge</i>	
An Evaluation Study of Architectural Design Decision Paradigms in Global Software Development	158
<i>Meiru Che and Dewayne Perry</i>	
An approach for classifying design artifacts (S)	164
<i>Sebastien Adam, Ghizlane El Boussaidi and Alain Abran</i>	

Data Mining for Knowledge Engineering

A Novel Hybrid Approach for Diarrhea Prediction	168
<i>Yongming Wang and Junzhong Gu</i>	
Are We Living in a Happy Country: An Analysis of National Happiness from Machine Learning Perspective (S)	174
<i>Theresia Ratih Dewi Saputri and Seok-Won Lee</i>	
BiBinConvmean : A Novel Biclustering Algorithm for Binary Microarray Data (S)	178
<i>Haifa Ben Saber and Mourad Elloumi</i>	

Intelligent Transportation Systems

Integration testing criteria for mobile robotic systems	182
<i>Maria Brito, Marcos Santos, Paulo Souza and Simone Souza</i>	
Embedded Real Time Blink Detection System for Driver Fatigue Monitoring	188
<i>Soheil Salehian and Behrouz Far</i>	
A Smartphone-based System for Automated Congestion Prediction (S)	195
<i>Lance Fiondella, Swapna Gokhale and Nick Lownes</i>	

Database and Information Systems

Endowing NoSQL DBMS with SQL Features Through Standard Call Level Interfaces	201
<i>Oscar Pereira, David Simões and Rui Aguiar</i>	

Optimizing of an Object-Oriented File System (OOFS) (S).....	208
<i>Ling-Hua Chang and Sanjiv Behl</i>	
An Evolution Mechanism for Dynamic Physical Applications in the Internet of Things (S)	213
<i>Kaibin Xie, Haiming Chen, Dong Li and Li Cui</i>	
<hr/> Software Product and Process Line <hr/>	
Architectural Evolution of a Software Product Line: an experience report	217
<i>Marcelo Laser, Elder Macedo Rodrigues, Anderson Domingues, Flavio Oliveira and Avelino F. Zorzo</i>	
Quality Evaluation of Artifacts in Tailored Software Process Lines (S)	223
<i>Camila Brondani, Gelson Bertuol and Lisandra Fontoura</i>	
BPMN* - A Notation for Representation of Variability in Business Process Towards Supporting Business Process Line Modeling (S)	227
<i>Marcelo Terenciani, Debora Paiva, Geraldo Landre and Maria Istela Cagnin</i>	
An Architecture Description Language for Dynamic Service-Oriented Product Lines (S) ..	231
<i>Seza Adjoyan and Abdelhak Seriai</i>	
<hr/> Social Networks <hr/>	
Social Analysis of the SEKE Co-Author Network	237
<i>Swapna Gokhale and Rehab El-Kharboutly</i>	
A Rule-based Method for Discovering Trajectory Profiles	244
<i>Lucas Andre Alencar, Luis Otavio Alvares, Chiara Renso, Alessandra Raffaeta and Vania Bogorny</i>	
A Balanced Method for Budgeted Influence Maximization	250
<i>Xinhui Xu, Yong Zhang, Qingcheng Hu and Chunxiao Xing</i>	
Using implications from FCA to represent a two mode network data (S)	256
<i>Sebastiao M. Neto, Mark A. J. Song, Luiz E. Zarate and Sergio M. Dias</i>	
<hr/> Programming Languages and Software Prototyping <hr/>	
How do developers use C++ libraries? An empirical study	260
<i>Di Wu, Lin Chen, Yuming Zhou and Baowen Xu</i>	
A Case Study Approach: Iterative Prototyping Model Based Detection of Macular Edema in Retinal OCT Images (S)	266
<i>Sadaf Sahar, Sadaf Ayaz and M.Usman Akram</i>	
A metrics-based comparative study on object-oriented programming languages	272
<i>Di Wu, Lin Chen, Yuming Zhou and Baowen Xu</i>	
TAGGINGSENSE: Method Based On Sensemaking For Object-Oriented Source Code Comprehension (S)	278
<i>Daniel Schreiber, Andre Menolli, Sheila Reinehr and Andreia Malucelli</i>	
<hr/> Software and Knowledge Visualization <hr/>	

Facilitating Peer Learning and Knowledge Sharing in STEM Courses via Pattern Based Graph Visualization.....	284
<i>Emilio Zegarra, Shi-Kuo Chang and Jingtao Wang</i>	
Scaffolding MATLAB and Octave Software Comprehension Through Visualization (S)....	290
<i>Ivan Lessa, Glauco Carneiro, Miguel Monteiro and Fernando Abreu</i>	
To Enlighten Hidden Facts in The Code: A Review of Software Visualization Metaphors (S).....	294
<i>Yangyang Xu, Yan Liu and Jiabin Zheng</i>	

Software Quality and Reliability

Reliability-Based Software Rejuvenation Scheduling for Cloud-Based Systems.....	298
<i>Jean Rahme and Haiping Xu</i>	
Reporting an Experience on the Establishment of a Quality Model for Systems-of-Systems (S).....	304
<i>Daniel Soares Santos, Brauner R. N. Oliveira, Adolfo Duran and Elisa Yumi Nakagawa</i>	
Experimental Frame Design Using E-DEVSML for Software Quality Evaluation (S).....	310
<i>Bei Cao, Huang Linpeng and Jianpeng Hu</i>	

Software Project Management II

Analysis of Risk Dependencies in Collaborative Risk Management(S).....	314
<i>Catherine Barchet, Luís Alvaro Silva and Lisandra Fontoura</i>	
A Practical Approach to Software Continuous Delivery Focused on Application Lifecycle Management (S).....	320
<i>Everton Gomedé, Rafael Thiago Da Silva and Rodolfo Miranda de Barros</i>	
Towards Effective Developer Recommendation in Software Crowdsourcing (S).....	326
<i>Shixiong Zhao, Beijun Shen, Yuting Chen and Hao Zhong</i>	

Requirements Engineering II

Creating User Scenarios through User Interaction Diagrams by Non-Technical Customers.	330
<i>Douglas Hiura Longo and Patrícia Vilain</i>	
How Stakeholders' Commitment May Affect the Success of Requirements Elicitation.....	336
<i>Corentin Burnay, Ivan Jureta and Stéphane Faulkner</i>	
An Exploration of System Architecture on Integrating Building Management System in High-Rise Building (S).....	342
<i>Zunhe Liu and Yan Liu</i>	

Testing II

Analyzing Exceptions in the Context of Test Data Generation Based on Symbolic Execution.....	346
<i>Marcelo Medeiros Eler, Vinicius Durelli and André Takeshi Endo</i>	

Automatically Evaluating the Efficiency of Search-Based Test Data Generation for Relational Database Schemas	352
<i>Cody Kinner, Gregory Kapfhammer, Phil McMinn and Chris Wright</i>	
Similarity-based regression test case prioritization (S)	358
<i>Rongcun Wang, Shujuan Jiang and Deng Chen</i>	
<hr/> Software Safety and Security I <hr/>	
Secure, Dynamic and Distributed Access Control Stack for Database Applications	364
<i>Oscar Pereira, Diogo Regateiro and Rui Aguiar</i>	
Specifying and Dynamically Monitoring the Exception Handling Policy	370
<i>Joilson Abrantes and Roberta Coelho</i>	
DefDroid: Securing Android with Fine-Grained Security Policy (S)	375
<i>Chao Huang, Shuohong Wang, Haiyang Sun and Zhengwei Qi</i>	
<hr/> Data Mining for Software Engineering <hr/>	
Developers' importance from the leader perspective	379
<i>Guilherme Tangari and Marcelo Maia</i>	
Stability of Three Forms of Feature Selection Methods on Software Engineering Data	385
<i>Huanjing Wang, Taghi Khoshgoftaar and Amri Napolitano</i>	
Building a Large-scale Software Programming Taxonomy from Stackoverflow	391
<i>Jiangang Zhu, Beijun Shen, Xuyang Cai and Haofen Wang</i>	
<hr/> Empirical Software Engineering II <hr/>	
An empirical study on predicting defect numbers	397
<i>Mingming Chen and Yutao Ma</i>	
Causes of Architecture Changes: An Empirical Study through the Communication in OSS Mailing Lists	403
<i>Wei Ding, Peng Liang, Antony Tang and Hans Van Vliet</i>	
A Behavior Marker tool for measurement of the Non-Technical Skills of Software Professionals: An Empirical Investigation	409
<i>Lisa Lacher, Gursimran Walia, Fabian Fagerholm, Max Pagels, Kendall Nygard and Jürgen Münch</i>	
A Platform for Empirical Research on Information System Evolution	415
<i>Robert Heinrich, Stefan Gärtner, Tom-Michael Hesse, Thomas Ruhroth, Ralf Reussner, Kurt Schneider, Barbara Paech and Jan Jürjens</i>	
<hr/> Software Assurance <hr/>	
A JVM-based Testing Harness for Improving Component Testability	421
<i>Weifeng Xu and Omar Ariss</i>	
Detecting Reporting Anomalies in Streaming Sensing Systems	427
<i>Shiree Hughes, Yuheng Du and Jason Hallstrom</i>	

Fault-Based Testing of Combining Algorithms in XACML3.0 Policies	433
<i>Dianxiang Xu, Ning Shen and Yunpeng Zhang</i>	

Software Defect Prediction

Combining Feature Subset Selection and Data Sampling for Coping with Highly Imbalanced Software Data	439
<i>Kehan Gao, Taghi Khoshgoftaar and Amri Napolitano</i>	
A Software Defect-Proneness Prediction Framework: A new approach using genetic algorithms to generate learning schemes (S)	445
<i>Juan Murillo-Morera and Marcelo Jenkins</i>	
Using Time Series Models for Defect Prediction in Software Release Planning (S)	451
<i>James Tunnell and John Anvik</i>	

Semantic Web

An Ontology for Describing Security Events	455
<i>Hossein Fani and Ebrahim Bagheri</i>	
CARP: Correlation Based Approach for Researcher Profiling (S)	461
<i>Hassan Nouredine, Iman Jarkass, Hussein Hazimeh, Omar Abou Khaled and Elena Mugellini</i>	
APRImora: A Semantic Architecture for Patterns Reuse (S)	465
<i>Angélica Aparecida de Almeida Ribeiro, Jugurta Lisboa-Filho, Lucas Francisco Da Matta Vegi, Alcione de Paiva Oliveira, Regina Maria Maciel Braga Villela and Emílio José de S. Fonseca</i>	

Topic Mining and Specification Mining

Mining Universal Specification Based on Probabilistic Model	471
<i>Deng Chen, Yanduo Zhang, Rongcun Wang, Xun Li, Li Peng and Wei Wei</i>	
Topic Matching Based Change Impact Analysis from Feature on User Interface of Mobile Apps (S)	477
<i>Qiwen Zou, Xiangping Chen and Yuan Huang</i>	
Learning Folksonomies for Trend Detection in Task-Oriented Dialogues (S)	483
<i>Gregory Wanderley and Emerson Paraiso</i>	
Towards Automatic Requirements Elicitation from Feedback Comments: Extracting Requirements Topics Using LDA (S)	489
<i>Hitoshi Takahashi, Hiroyuki Nakagawa and Tatsuhiro Tsuchiya</i>	

User Experience and Organizational Learning

MAX: A Method for Evaluating the Post-use User eXperience through Cards and a Board	495
<i>Emanuelle Cavalcante, Luis Rivero and Tayana Conte</i>	
Designing Personas with Empathy Map (S)	501
<i>Bruna Ferreira, Williamson Silva, Edson Oliveira and Tayana Conte</i>	

An approach to identify relevant subjects for supporting the Learning Scheme creation task (S)	506
<i>Huander Tironi, Andre Menolli, Sheila Reinehr and Andreia Malucelli</i>	

Software Measurement and Metrics

Using peak analysis for identifying lagged effects between software metrics (S)	512
<i>Josee Tasse</i>	
Integration of Software Measurement Supporting Tools: A Mapping Study (S)	516
<i>Vinícius S. Fonseca, Monalessa P. Barcellos and Ricardo De A. Falbo</i>	
Toward using Business Process Intelligence to Support Incident Management Metrics Selection and Service Improvement (S)	522
<i>Bianca Trinkenreich, Gleison Santos, Valdemar Confort and Flavia Santoro</i>	

Software Safety and Security II

Study on the Accident-causing Model Based on Safety Region and Applications in China Railway Transportation System	528
<i>Yong Qin, Hui Ma, Miao Du and Limin Jia</i>	
Statically-Guided Fork-based Symbolic Execution for Vulnerability Detection (S)	536
<i>Yue Wang, Hao Sun and Qingkai Zeng</i>	
How Does Defect Removal Activity of Developer Vary with Development Experience? (S)	540
<i>Reou Ando, Seiji Sato, Chihiro Uchida, Hironori Washizaki, Yoshiaki Fukazawa, Sakae Inoue, Hiroyuki Ono, Yoshiiku Hanai, Masanobu Kanazawa, Kazutaka Sone, Katsushi Namba and Mikihiko Yamamoto</i>	

Model Transformation and Comparison

Model Comparison: a Systematic Mapping Study	546
<i>Lucian Gonçales, Kleinner Farias, Murillo Scholl, Toacy Oliveira and Mauricio Veronez</i>	
Exploring SOA Pattern Performance using Coupled Transformations and Performance Models	552
<i>Nariman Mani, Dorina Petriu and Murray Woodside</i>	
On the Specification of Model Transformations through a Platform Independent Approach (S)	558
<i>Ana Patricia Magalhaes, Aline Andrade and Rita Suzana Pitangueira Maciel</i>	

Testing III

Improved Metrics for Non-Classic Test Prioritization Problems (S)	562
<i>Ziyuan Wang</i>	
An Average Case Time Complexity Estimator for Black-box Functions	567
<i>Duncan Yung, Bill Laboon and Shikuo Chang</i>	

Automatic Detection of Parameter Shielding for Test Case Generation (S)	571
<i>Jingjian Lin, Jun Yan and Jifeng Xuan</i>	

Petri Nets and Formal Methods

PIPE+Verifier - A Tool for Analyzing High Level Petri Nets	575
<i>Su Liu and Xudong He</i>	
SANGE – Stochastic Automata Networks Generator. A tool to efficiently predict events through structured Markovian models (S)	581
<i>Joaquim Assunção, Paulo Fernandes, Lucelene Lopes, Angelika Studeny and Jean-Marc Vincent</i>	
Modeling and Analyzing Adaptive Energy Consumption for Service Composition (S)	585
<i>Guisheng Fan, Huiqun Yu and Liqiong Chen</i>	
Modeling and Analyzing Publish Subscribe Architecture using Petri Nets	589
<i>Junhua Ding and Dongmei Zhang</i>	

Program Verification

Flexible and Extensible Runtime Verification for Java	595
<i>Chengcheng Xiang, Zhengwei Qi and Walter Binder</i>	
Improving the Accuracy of Integer Signedness Error Detection Using Data Flow Analysis .	601
<i>Hao Sun, Chao Su, Yue Wang and Qingkai Zeng</i>	
Extracting More Object Usage Scenarios for API Protocol Mining	607
<i>Deng Chen, Yanduo Zhang, Rongcun Wang, Binbin Qu, Jianping Ju and Wei Wei</i>	

Domain-Specific Languages

NeoIDL: A Domain-Specific Language for Specifying REST Services	613
<i>Rodrigo Bonifacio, Thiago M. Castro, Ricardo Fernandes, Alisson Palmeira and Uirá Kulesza</i>	
A Unified MapReduce Domain-Specific Language for Distributed and Shared Memory Architectures	619
<i>Daniel Adornes, Dalvan Griebler, Cleverson Ledur and Luiz Gustavo Fernandes</i>	
Towards a Metamodel Design Methodology: Experiences from a model transformation metamodel design	625
<i>Ana Patricia Magalhaes, Rita Suzana Pitangueira Maciel and Aline Andrade</i>	

Web Applications and Ontological Engineering

Finding and Emulating Keyboard, Mouse, and Touch Interactions and Gestures while Crawling RIA's	631
<i>Frederik Nakstad, Hironori Washizaki and Yoshiaki Fukazawa</i>	
An Oracle based on Image Comparison for Regression Testing of Web Applications	639
<i>Akihiro Hori, Shingo Takada, Haruto Tanno and Morihide Oinuma</i>	

Towards the Anonymisation of RDF Data	646
<i>Filip Radulovic, Raúl García-Castro and Asunción Gómez-Pérez</i>	
An Information Retrieval Model using Query Expansion based on Ontologies in the Computer Science Domain (S)	652
<i>Bonnie G. Carranza Chávez and Andrés Melgar</i>	

Requirements Analysis and Software Architecture

Toward an Architecture for Model Composition Techniques (S)	656
<i>Kleinner Farias, Lucian Goncales, Murilo Scholl, Toacy Oliveira and Maurício Veronez</i>	
JSAN: A Framework to Implement Normative Agents (S)	660
<i>Marx L. Viana, Paulo Alencar, Everton T. Guimarães, Francisco J. P. Cunha, Donald Cowan and Carlos J. P. Lucena</i>	
Quantitative Reasoning of Goal Satisfaction in the i*Framework (S)	666
<i>Chitra M Subramanian, Aneesh Krishna, Arshinder Kaur and Raj P Gopalan</i>	
CQV-UML Tool: a tool for managing the impact of change on UML models (S)	670
<i>Dhikra Kchaou, Nadia Bouassida and Hanène Ben Abdallah</i>	
An evolution management model for multi-level component-based software architectures ..	674
<i>Abderrahman Mokni, Marianne Huchard, Christelle Urtado, Sylvain Vauttier and Huaxi Yulin Zhang</i>	

Empirical Software Engineering III

Using Learning Styles of Software Professionals to Improve their Inspection Team Performance	680
<i>Anurag Goswami, Gursimran Walia and Abhinav Singh</i>	
How do software engineers apply an early usability inspection technique? A qualitative study	686
<i>Natasha Valentim, Tayana Conte, Bernardo José and Rafael Prikladnicki</i>	
A Empirical Study on the Status of Software Localization in Open Source Projects (S) ...	692
<i>Zeyad Alshaikh, Shaikh Mostafa, Xiaoyin Wang and Sen He</i>	

Agile Software Development and User Interface

bibin-DA: An Agile Domain Analysis Process and its Industrial Evaluation (S)	696
<i>Tassio Vale, Iuri Souza, Ivonei Silva and Eduardo Almeida</i>	
A Feature-Based Tool-Selection Classification for Agile Software Development (S)	700
<i>Mohsen Taheri and S. Masoud Sadjadi</i>	
Adoption of Software Product Line Development to an Environment of Voice User Interface (S)	705
<i>Diógenes R. F. Oliveira, Byron L. D. Bezerra, Elyda L. S. X. Freitas and Alexandre M. A. Maciel</i>	

Adopting Agile Methods in the Public Sector: A Systematic Literature Review (S)	709
<i>Isaque Vacari and Rafael Prikladnicki</i>	

Poster/Demo

Modeling Framework for Developing and Testing Network Security Techniques against DDoS Attacks (P)	715
<i>Konstantin Borisenko, Ivan Kholod and Andrey Shorov</i>	
Natural Language Processing to Quantify Security Effort in the Software Development Lifecycle (P)	716
<i>Constantine Aaron Cois and Rick Kazman</i>	
Towards Goal-Oriented Conformance Checking (P)	722
<i>Hiroki Horita, Hideaki Hirayama, Yasuyuki Tahara and Akihiko Ohsuga</i>	
Image retrieval based on structural and textual context (P)	725
<i>Sana Fakhfakh Akrouf, Mohamed TMAR and Walid MAHDI</i>	
Probabilistic Failure-causing Schema in Input-Domain Testing (P)	726
<i>Ziyuan Wang</i>	
CARE: A Computer-Aided Requirements Engineering Tool for Problem-Oriented Software Development (P)	727
<i>Guoyuan Liu, Zhi Li and Zhaofeng Ouyang</i>	
ExpOse: Inferring Worst-case Time Complexity by Automatic Empirical Study (P)	730
<i>Cody Kinneer, Gregory Kapfhammer, Chris Wright and Phil McMinn</i>	
Modeling China Metro Train Route Occlusion Operation Method Based on Time Petri Nets (P)	732
<i>Ye Zhang</i>	
Modeling China Metro Train Route Occlusion Operation Method Based on Time Petri Nets (D)	735
<i>Ye Zhang and Yatao Wang</i>	
System Architecture of a Train Sensor Network for Ubiquitous Safety Monitoring (P)	736
<i>Guoqiang Cai and Mengchu Zhou</i>	
System Architecture of a Train Sensor Network for Ubiquitous Safety Monitoring (D)	739
<i>Guoqiang Cai</i>	
Agile Practices in Maturity Model for Testing: an Experience Report (D)	740
<i>Ana Paula Cavalcanti Furtado, Suzana Sampaio, Ermeson Andrade, Ivaldir Junior and Marcos André Wanderley Gomes</i>	

Author's Index	A-1
Program Committee Reviewers' Index	A-11
External Reviewers' Index	A-16

Note:

(S) indicates a short paper.

(P) indicates a poster, which is not a refereed paper.

(D) indicates a demo.

Towards Building Knowledge on Causes of Critical Requirements Engineering Problems

Marcos Kalinowski
UFF
Niterói, Brazil
kalinowski@ic.uff.br

Rodrigo Oliveira Spínola
UNIFACS/Fraunhofer
Salvador, Brazil
rodrigo.spinola@pro.unifacs.br

Tayana Conte
UFAM
Manaus, Brazil
tayana@icomp.ufam.edu.br

Rafael Prikladnicki
PUC-RS
Porto Alegre, Brazil
rafael.prikladnicki@puers.br

Daniel Méndez Fernández
Technische Universität München
München, Germany
daniel.mendez@tum.de

Stefan Wagner
University of Stuttgart
Stuttgart, Germany
stefan.wagner@informatik.uni-stuttgart.de

Abstract—[Context] Many software projects fail due to problems in requirements engineering (RE). [Objective] The goal of this paper is to gather information on relevant RE problems and to represent knowledge on their most common causes. [Method] We replicated a global family of RE surveys in Brazil and used the data to identify critical RE problems and to build probabilistic cause-effect diagrams to represent knowledge on their common causes. [Results] The survey was answered by 74 different organizations, including small, medium and very large sized companies, conducting both, plan-driven and agile development. The most critical RE problems, according to those organizations, are related to communication and to incomplete or underspecified requirements. We provide the full probabilistic cause-effect diagrams with knowledge on common causes of the most critical identified RE problems online. [Conclusion] We believe that the knowledge presented in the diagrams can be helpful to support organizations in conducting causal analysis sessions by providing an initial understanding on what usually causes critical RE problems.

Keywords—Survey; NaPiRE; Knowledge Building; Requirements Engineering; Problems; Causes; Causal Analysis.

I. INTRODUCTION

The importance of high-quality requirements engineering (RE) has been widely accepted and well documented. Pfleeger [1] states that efficient RE is one of the main factors to avoid software project failure. RE constitutes a holistic key to successful development projects [2]. However, industry is still struggling to apply high-quality RE practices [3] and getting a further understanding on common RE problems and their causes is of great interest to both industry and academy. Therefore, many researchers have addressed identifying and analyzing RE problems faced by industry [4][5].

More recently, a project called NaPiRE (Naming the Pain in Requirements Engineering) comprises the design of a family of surveys on RE practice and problems, and it is conducted in joint collaboration with various researchers from different countries [6][7]. The goal of this project is to lay an empirical foundation

about the state of the practice in RE to allow steering future research in a problem-driven manner [6]. Currently, the NaPiRE survey is being conducted in several countries around the globe.

Conducting causal analysis sessions [8] is an efficient means for organizations to improve their practice to overcome problems faced during software development. In these sessions, the causes of problems are identified and addressed to prevent their recurrence in future projects.

Experience reports on conducting causal analysis sessions on RE problems can be found in [9], [10] and [11]. One of the main difficulties reported during those sessions concerns the absence of a starting point for identifying potential causes. An initial solution concept to address this problem has been proposed in [12], where an approach for integrating knowledge of successive causal analysis sessions is described. This approach introduced the concept of a probabilistic cause-effect diagram, and of using such diagrams to present accumulated knowledge on the probabilities of causes based on the organization's prior causal analysis experiences on similar problems.

However, although this approach and the probabilistic cause-effect diagram showed to be useful to support causal analysis sessions in a proof of concept [13], an experimental study [14] and an industrial experience [9], the knowledge depicted in the diagram has to be generated based on intra-company data from previous causal analysis sessions. Thus, it has to be built gradually and from scratch for each context, as there is no general documented and empirically grounded knowledge causes of critical problems that could be used as a starting point.

In this paper, we aim at gathering information on relevant RE problems and to represent knowledge on their most common causes as reported by the industry. Therefore, we replicated the NaPiRE survey in Brazil. We got answers from 74 different Brazilian organizations, spread across the country. We then used the data to identify the reportedly most critical RE problems and organized knowledge on their common causes by building

probabilistic cause-effect diagrams for those RE problems and making them available online¹. Therefore, we enable organizations to use the knowledge presented in these diagrams as a starting point when conducting causal analysis by providing a further understanding on common causes of RE problems.

As an initial evaluation, we interviewed an industry representative of a Brazilian CMMI-Dev level 3 company who is currently implementing causal analysis practices, showing her the probabilistic cause-effect diagrams. The feedback was positive and future work includes conducting a case study of using those diagrams in industry while conducting causal analysis sessions on RE problems.

The remainder of this paper is organized as follows. Section II describes related work. Section III describes the NaPiRE project and its replication in Brazil. Section IV presents the survey results on the most critical RE problems and the probabilistic cause-effect diagrams with knowledge on common causes of those problems. Section V discusses the obtained results and their limitations in the light of the diagrams and of an informal interview conducted with an industry representative. Section VI presents the concluding remarks and future work.

II. RELATED WORK

In this paper, we aim at identifying relevant RE problems and building knowledge on their common causes, by replicating a survey. We propose representing such knowledge using probabilistic cause-effect diagrams. The following subsections provide the related work on survey research on RE problems and on probabilistic cause-effect diagrams.

A. Survey Research on RE Problems

Well-known surveys on causes for project failure include the Chaos Report of the Standish Group on cross-company root causes for project failures. While most of these causes are related to RE, the survey has serious design flaws and the validity of its results is questionable [15]. Moreover, it exclusively investigated failed projects and general causes at the level of overall software processes. Thus, it does not directly support the investigation of RE problems in industry.

Some surveys have been focusing specifically on RE problems in industry. These surveys include the one conducted by Hall *et al.* [4] in twelve software organizations. Their findings, among others, suggest that most RE problems are organizational rather than technical.

Some country-specific investigations of RE problems include the surveys conducted by Solemon *et al.* [16] and Liu *et al.* [17], with Malaysian and Chinese organizations, respectively. Khankaew and Riddle [5], more recently conducted semi-

structured interviews with organizations from Thailand. In the first results of the NaPiRE survey, the reported RE problems were mainly identified from German companies [6].

These investigations provide valuable insights into industrial environments. However, as each of them focuses on specific aspects in RE or on specific countries, their results are isolated and not generalizable. To address this issue, the NaPiRE survey was designed in a joint collaboration as a continuous research project with researchers from different countries [6]. The design and interpretation of the results are aligned to a theory [6]. The survey, being replicated in different countries, shall contribute to an empirical basis to allow generalizable and problem-driven research in RE [6].

Given this context, to gather data concerning RE problems, the Brazilian authors of this paper decided to join the NaPiRE team and to replicate this survey in Brazil. To facilitate the use of this knowledge on common causes of RE problems it was organized into probabilistic cause-effect diagrams [12]. More details on these diagrams follow.

B. Probabilistic Cause-Effect Diagrams

Probabilistic cause-effect diagrams were introduced in [12] to provide visual support in causal analysis sessions with knowledge on common causes of problems gathered from previous experiences. They have shown to be a useful instrument in a proof of concept [13], an experimental study [14], and an industrial experience [9].

An example of such a diagram, taken from the experience reported in [9] is shown in Figure 1. The diagram extends the traditional cause-effect diagram [18] by (a) showing the probabilities for each possible cause to lead to the analyzed problem, and (b) representing the causes using grey tones, where causes with higher probability are shown closer to the center and in darker tones. Following the suggestion of guidelines for conducting causal analysis [8], it organizes the causes of problems into five categories: Input, Method, Organization, People, and Tool. The probabilities shown in Figure 1 were calculated with data on causes gathered in successive causal analysis sessions conducted in earlier iterations of the project. Causes that happened more frequently have higher probabilities.

This representation can be easily interpreted by causal analysis teams and highlights causes with greater probabilities of creating the analyzed problem. It allows the teams to efficiently answer questions during causal analysis sessions, such as: “Given similar past projects within my organizational context, with which probability does a certain cause lead to a specific problem?”. During the causal analysis sessions, the team can use the probabilistic cause-effect diagram, together with data on the problem, as input to help building a new cause-effect diagram with the causes identified in the current session. The newly identified causes can then be used to update the probabilities for

¹ <http://www.ic.uff.br/~kalinowski/seke15>

the next session. This support has shown to be useful to support efficient cause identification [9][14].

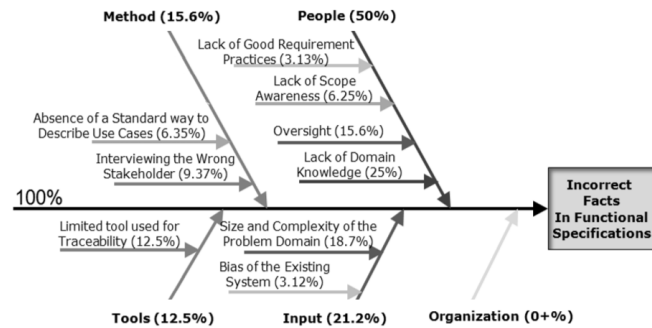


Figure 1. A probabilistic cause-effect diagram based on intra-company industry data for incorrect facts in functional specifications, taken from [9].

However, the main shortcoming of using these diagrams is that the knowledge on causes of problems is generated intra-company and has to be built gradually. We believe that cross-company data taken from a sufficiently wide range of data from industry, with knowledge on causes of problems provides useful initial input when analyzing those problems. The problems have to be calibrated later with intra-company data containing the specific causes identified in new causal analysis sessions.

To build the necessary inter-company knowledge on common causes of critical RE problems, initially based on data from Brazilian companies, we replicated the NaPiRE survey in Brazil. Information on the NaPiRE project and on the conducted replication in Brazil is described next.

III. NAPIRE BRAZIL

A. The NaPiRE Project

The NaPiRE project resulted in the design of a global family of surveys to overcome the problem of isolated investigations in RE that are not representative [6]. Thus, a long-term goal of the project is to establish an empirically sound basis for understanding trends and problems in RE [7].

The design of the survey and its instruments have been extensively reviewed by several researchers [6]. In summary, the NaPiRE survey contains 35 questions gathering the following type of data from the responding organizations: (a) general information, (b) RE status quo, (c) RE improvement status quo, (d) RE problems faced in practice, and (e) RE problem manifestation (e.g., causes, impact).

The family of surveys is currently being conducted in several countries. Further information on the project, including the countries in which the survey is being replicated and a sample of

the questionnaire can be found online². Concerning its results, so far initial results from Germany have already been published [6].

B. NaPiRE Survey Replication in Brazil

When we decided to replicate the NaPiRE survey in Brazil, it was already designed and all the instruments were available. Therefore, in this section we focus on the details of how we planned and operated the replication in Brazil. Further information on the design of the surveys can be found in [6].

To plan the survey replication in Brazil, we held a couple of meetings with the NaPiRE general organizers². During these meetings, the online environment (EFS survey tool³) was presented and some general guidelines for conducting the survey were provided. We decided to translate all instruments to Portuguese, the participants' native language.

Given the geographic dimensions of Brazil, to reach organizations from different regions and to gather representative data, the first author assembled a team of industry-focused researchers spread across the country. The strategy consisted of having researchers from the four main industry intensive regions of the country involved. The resulting NaPiRE Brazil team² comprises a researcher from the South of the country, one from the Southeast, one from the North and one from the Northeast.

Additionally, we contacted Softex, the organization responsible for the most widely adopted software reference model in Brazil, the MPS-SW, with over 600 assessments in all Brazilian regions [19]. They promptly trusted us contacts of 254 organizations with currently valid MPS-SW assessments so that they could be invited to take part in the survey.

Including a set of 80 additional relevant industry contacts from the authors (20 contacts per author on average), we created a list with contacts of representatives from 334 software organizations. We believe this set to be representative for the Brazilian software industry. Given the size of this industry (thousands of software organizations [20]), an extensive survey to reach all of them would be almost impossible.

We then configured the environment and sent the invitations with a link and password to the online survey to the list of contacts by e-mail. The survey was sent in December 2014, with reminders in January 2015 and February 2015. In total, 118 of the 334 invited organization representatives logged in to answer the survey. Out of these, 74 representatives answered the questionnaire completely (9 only read the initial instructions, 18 dropped at the first page of the questionnaire, and 17 dropped the survey in the middle). The median time to answer the survey completely was 29 minutes.

² www.re-survey.org

³ www.unipark.com/en

IV. TOWARDS BUILDING KNOWLEDGE ON CAUSES OF CRITICAL RE PROBLEMS

In this section, we provide the initial survey results concerning the identified critical RE problems and their common causes as reported by industry. We also explain how the gathered information was organized into probabilistic cause-effect diagrams to provide a further understanding on common causes of RE problems. We start by presenting the characterization of the responding organizations as this information is crucial to enable a correct interpretation of the results.

A. Characterization of the Responding Organizations

To provide a summary of the characterization of the responding organizations, we will present information on their size and the used process models and RE standards. We will also present the roles of the participants within the organizations and their experience in this role.

Concerning size, in Table I we can observe that the survey included both extremes, small and medium-sized and very large-sized organizations.

TABLE I. SIZE OF THE SURVEYED ORGANIZATIONS

Size*	No. of Answers
1-10 Employees	11 (15.49%)
11-50 Employees	15 (21.13%)
51-250 Employees	17 (23.94%)
251-500 Employees	5 (7.04%)
501-1000 Employees	3 (4.23%)
1001-2000 Employees	5 (7.04%)
More than 2000 Employees	15 (21.13%)
Invalid (missing) answers	3 (N/A)

* Size including software and other areas.

Regarding the process model, Table II shows that most of the surveyed organization adopt agile (mainly Scrum-based) process models, followed by iterative and incremental process models and the traditional waterfall model. It is noteworthy that some organizations informed to use more than one process model to handle different types of projects. One explanation for changing process models is that organizations might have to follow a waterfall model during a bidding procedure while adopting scrum once the project is formally assigned.

TABLE II. PROCESS MODEL

Process Model	No. of Answers
Scrum	45 (60.81%)
Waterfall	22 (29.73%)
Rational Unified Process (RUP)	19 (25.68%)
Extreme Programming (XP)	7 (9.46%)
V Model	4 (5.41%)
Others*	11 (14.86%)

* Others includes self-adapted process models (4), other iterative and incremental development process models (4) and other process models based on agile methods (3).

In Table III, we can observe that most of the surveyed organizations follow reference-model-based standards, such as MPS-SW and CMMI-Dev. This, of course, may have been influenced by the strategy of also distributing the survey to the organizations with valid MPS-SW assessments. Nevertheless, many organizations answered that they follow the standards of the adopted development process and their own standards.

TABLE III. RE STANDARD (OR REFERENCE MODEL)

RE Standard	No. of Answers
SW reference model (e.g., CMMI-Dev, MPS-SW)	39 (52.70%)
Adopted development process (e.g., RUP, Scrum)	25 (33.78%)
Self-defined (including a process with deliverables, milestones and phases)	19 (25.68%)
Self-defined (including a process with roles and responsibilities)	18 (24.32%)
Self-defined (including artefacts and templates)	18 (24.32%)
None	1 (1.35%)

To characterize the participants, their roles in the organization are shown in Table IV and their experience in these roles is shown in Table V. It can be seen that participants are mainly project managers and highly experienced.

TABLE IV. ROLES OF THE PARTICIPANTS

Role	No. of Answers
Project Manager	32 (45.07%)
Business Analyst	8 (11.27%)
Developer	4 (5.63%)
Software Architect	4 (5.63%)
Test Manager / Tester	3 (4.23%)
Requirements Engineer	2 (2.82%)
Others*	18 (25.35%)
Invalid (missing)	3 (NA)

* Other informed values include development directors, program managers and portfolio managers (7), quality assurance analysts (7), and people from the software engineering process group (4).

TABLE V. EXPERIENCE OF PARTICIPANTS IN THEIR ROLES

Experience	No. of Answers
Specialist (more than 3 years)	52 (73.24%)
Experienced (1 to 3 years)	15 (21.13%)
Newbie (up to 1 year)	04 (5.63%)

While we had no control over which organizations and representatives would answer the survey, we were happy to obtain such a representative characterization, including small, medium and very large-sized organizations enrolled in both, plan-driven and agile development methods, and to have our answers provided mainly by highly experienced professionals.

B. Critical RE Problems

Based on a set of 21 precompiled general RE problems listed in the NaPiRE questionnaire [6], the participants were asked to rank the five most critical ones.

The most critical RE problems, as ranked by the survey participants, are shown in Table VI. This table shows the 8 RE problems that were cited between the five most critical ones by more than 20% of the respondents. The table also shows how often each of these problems was ranked as being the most critical problem of all. We observe that communication problems were often cited (problems #1 and #4), as well as incomplete and underspecified requirements (problems #2 and #3).

TABLE VI. MOST CRITICAL RE PROBLEMS

#	RE Problems	Cited*	Ranked #1*
1	Communication flaws between the project team and the customer	32 (43.24%)	9 (12.16%)
2	Incomplete and/or hidden requirements	31 (41.89%)	12 (16.22%)
3	Underspecified requirements that are too abstract and allow for various interpretations	31 (41.89%)	3 (4.05%)
4	Communication flaws within the project team	26 (35.14%)	5 (6.67%)
5	Insufficient support by customer	21 (28.38%)	5 (6.76%)
6	Inconsistent requirements	18 (24.32%)	2 (2.70%)
7	Time boxing / Not enough time in general	17 (22.97%)	1 (1.35%)
8	Moving targets (changing goals, business processes and/or req.)	15 (20.27%)	5 (6.67%)

* The probabilities were calculated based on the overall amount of 74 participants although some of them (9) did not inform any of the problems. We decided to keep the total amount as basis because we were not sure if they did not find the problems relevant or if they did not want to think about it.

C. Causes of Critical RE Problems

After selecting the five most critical RE problems, we asked our respondents to provide what they believe of being the main causes for each of the problems. They provided the causes in an open question format, with one open question for each of the previously selected RE problems.

We analyzed the provided qualitative data and aggregated similar causes even when textual descriptions differed; always counting the number of times each cause was reported for a given problem. Therefore, we used the constant comparative method [21] to compare each textual cause description against our already catalogued list of causes.

Thereafter, to build the probabilistic cause-effect diagrams for each RE problem, we categorized the causes as suggested in [8] and generated probabilities based on frequency counting. Figure 2 shows the probabilistic cause-effect diagram for the problem ‘Incomplete and/or hidden requirements’ (#2 in Table VI). The 31 organizations that ranked this problem among the most critical provided 34 instances of causes for it, which could be mapped to a list of 20 catalogued causes with different frequencies that were used as input to generate the diagram. It can be seen that, according to the survey participants, the most common causes for this problem are related to the *people* category and the *lack of skills in RE* (29.41%), although the

Method and *Input* categories each were responsible for more than 20% of the causes informed for this problem.

Due to space limitations, the remaining probabilistic cause-effect diagrams for the five top ranked RE problems are available online¹, where the user can select the problem and then look at the respective diagram to obtain a further understanding on common causes.

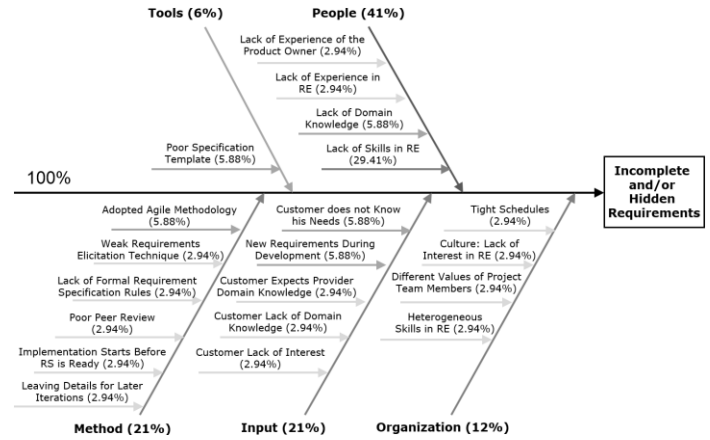


Figure 2. Probabilistic cause-effect diagram based on problem ‘Incomplete and/or hidden requirements’ based on the surveyed industry data.

V. DISCUSSION

Given our previous causal analysis experiences, the wide range of organizations that participated in the survey and their representative characterization (including, for instance, differently sized plan-driven and agile-oriented organizations), we believe that the resulting probabilistic cause-effect diagrams generated from cross-company data should provide useful additional input into causal analysis sessions for companies working in same or similar project settings. This is especially true for organizations that do not have any accumulated knowledge on common causes of RE problems.

It is noteworthy that organization may also need to calibrate the probabilities of the diagrams with the causes identified in their own causal analysis sessions, by using an approach similar to the one detailed in the experience described in [9]. Still, we believe that the knowledge on common causes generated as a contribution of this paper constitutes a useful starting point.

To provide some preliminary support for these claims, we interviewed an industry representative of the software engineering process group of a Brazilian CMMI-Dev level 3 company that is currently implementing causal analysis practices. We showed her the probabilistic cause-effect diagrams and she found the contained knowledge useful and was promptly willing to use the diagrams to support causal analysis sessions in her environment. This strengthens our confidence in the suitability of our results to establish an intra-company knowledge base on common RE problems and their causes.

VI. CONCLUDING REMARKS

In this paper, we gathered data on critical RE problems and their common causes by replicating the NaPiRE survey in Brazil. We presented the results concerning the most critical RE problems and represented the knowledge on common causes of these problems by building probabilistic cause-effect diagrams.

The chosen dissemination strategy enabled us to get answers from a wide range (74) of Brazilian organizations. The characterization showed a large diversity of the responding organizations including differently sized plan-driven and agile-oriented organizations.

The survey results allowed us to identify the most critical RE problems according to the responding organizations (Table VI) and to observe that they are mainly related to communication problems and incomplete or underspecified requirements. In addition, the probabilistic cause-effect diagrams (see Figure 2) showed to be suitable for the presentation of knowledge on common causes of the RE problems in an easily understandable way. The probabilistic cause-effect diagrams for the RE problems identified as the five most critical ones are available online¹. We believe that these diagrams provide useful input into causal analysis sessions at specific organizations, especially for organizations that do not have accumulated knowledge on common causes of their RE problems. Case studies on this matter form a high-priority scope of our future work.

Future work also comprises: (a) considering more NaPiRE data (from different countries) to build the knowledge on common causes based on a larger cross-company dataset, (b) further exploring other data gathered as part of the NaPiRE Brazil survey, such as RE problem mitigation actions, and (c) integrating the overall survey results into an empirical software engineering body of knowledge [22].

ACKNOWLEDGMENT

The authors would like to thank Softex and each of the 74 responding organizations. Thanks also to CNPq for financial support (project #460627/2014-7).

REFERENCES

- [1] S.L. Pfleeger, "Software Engineering: Theory and Practice", 4th edition, Prentice-Hall, 2009.
- [2] M. Broy, "Requirements Engineering as a Key to Holistic Software Quality", In: International Symposium on Computer and Information Sciences (ISCIS 2006), pp. 24–34, 2006.
- [3] D. Méndez Fernández, S. Wagner, K. Lochmann, A. Baumann, H. de Carne, "Field Study on Requirements Engineering: Investigation of Artefacts, Project Parameters, and Execution Strategies", Information and Software Technology, vol. 54, pp. 162–178, 2012.
- [4] T. Hall, S. Beecham and A. Rainer, "Requirements Problems in Twelve Software Companies: an Empirical Analysis", Empirical Software Engineering, vol. 8, pp. 7–42, 2003.
- [5] S. Khankaew and S. Riddle, "A review of practice and problems in requirements engineering in small and medium software enterprises in

- Thailand", In: International Workshop on Empirical Requirements Engineering (EmpiRE), pp.1-8, 2014.br
- [6] D. Méndez Fernández and S. Wagner, "Naming the Pain in Requirements Engineering: A Design for a Global Family of Surveys and First Results from Germany", Information and Software Technology, vol. 57, pp. 616-643, 2015.
- [7] D. Méndez Fernández and S. Wagner, "Naming the pain in requirements engineering: Design of a global family of surveys and first results from Germany", in: International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. 183–194, 2013
- [8] M. Kalinowski, D.N. Card and G.H. Travassos, "Evidence-Based Guidelines to Defect Causal Analysis", IEEE Software, Vol. 29, Issue 4, pp. 16-18, 2012.
- [9] M. Kalinowski, E. Mendes and G.H. Travassos, "An Industry Ready Defect Causal Analysis approach exploring Bayesian Networks", In: Software Quality Days, pp 12-33, 2014.
- [10] O. Kovalenko, D. Winkler, M. Kalinowski, E. Serral and S. Biffi, "Engineering Process Improvement in Heterogeneous Multi-Disciplinary Environments with Defect Causal Analysis", In: European Conference on System, Software & Service Process Improvement (EuroSPI), 2014.
- [11] M. Kalinowski, R.O. Spínola, A.C. Dias-Neto, A. Bott and G.H. Travassos, "Inspeções de Requisitos de Software em Desenvolvimento Incremental: Uma Experiência Prática", In: VI Simpósio Brasileiro de Qualidade Software (SBQS), Porto de Galinhas, Brazil, 2007.
- [12] M. Kalinowski, G.H. Travassos and D.N. Card, "Towards a Defect Prevention Based Process Improvement Approach", In: Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 199-206, 2008.
- [13] M. Kalinowski, G.H. Travassos, E. Mendes and D.N. Card, "Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks", In: International Conference on Product Focused Software Development and Process Improvement (PROFES), pp. 92–106, 2010.
- [14] M. Kalinowski, E. Mendes, and G.H. Travassos, "Automating and Evaluating the Use of Probabilistic Cause-Effect Diagrams to Improve Defect Causal Analysis", In: International Conference on Product Focused Software Development and Process Improvement (PROFES), pp. 232-246, 2011.
- [15] J. Eveleens and T. Verhoef, "The Rise and Fall of the Chaos Report Figures", IEEE Software, vol. 27, pp. 30-36, 2010.
- [16] B. Solemon, S. Sahibuddin and A. A. Abd Ghani, "Requirements Engineering Problems and Practices in Software Companies: An Industrial Survey", Advances in Software Engineering, vol. 59, pp.70-77, 2009.
- [17] L. Liu, T. Li and F. Peng, "Why requirements engineering fails: A survey report from china", International Conference on Requirements Engineering (RE), pp. 317-322, 2010.
- [18] K. Ishikawa, "Guide to Quality Control", Asian Productivity Organization, Tokyo, 1976.
- [19] M. Kalinowski, K. Weber, N. Franco, V. Duarte, G. Santos, and G. Travassos, "Results of 10 Years of Software Process Improvement in Brazil Based on the MPS-SW Model", In: Int. Conf. on the Quality in Information and Communications Technology (QUATIC), pp.28-37, 2014.
- [20] Softex, "Software e Serviços de TI: A Indústria Brasileira em Perspectiva", Observatório Softex (ISSN 1984-6797), vol. 2, 2012.
- [21] C. B. Seaman, "Qualitative methods in Empirical Studies of Software Engineering", IEEE Transactions on Software Engineering (TSE), vol. 25, no. 4, pp. 557-572, 1999.
- [22] S. Biffi, M. Kalinowski, R. Rabiser, F.J. Ekaputra and D. Winkler, "Systematic Knowledge Engineering: Building Bodies of Knowledge from Published Research", International Journal on Software Engineering and Knowledge Engineering (IJSEKE), vol. 24, No. 10, pp. 1–39, 2014.

Identification and Classification of Requirements from App User Reviews

Hui Yang

State Key Lab of Software Engineering
School of Computer, Wuhan University, China
huiyang@whu.edu.cn

Peng Liang*

State Key Lab of Software Engineering
School of Computer, Wuhan University, China
liangp@whu.edu.cn

Abstract—Review function, as a feedback mechanism from users to developers and vendors, is provided by most APP distribution platforms that allow users to rate and comment an APP after using it. User reviews are recognized as a valuable source to improve APPs and increase the value for users. With the sharp increase in the amount of user reviews, how to effectively and efficiently analyze the user reviews and identify potential and critical user needs from them to improve the APPs becomes a challenge. In this paper, we propose an approach to automatically identify requirements information and further classify them into functional and non-functional requirements from user reviews, using a combination of information retrieval technique (TF-IDF) and NLP technique (regular expression) with human intervention in keywords selection for requirements identification and classification. We validated the proposed approach with the user reviews collected from a popular APP iBooks in English App Store, and further investigated the cost and return of our approach: how the size of sample reviews for keywords selection (cost) affects the classification results in precision, recall, and F-measure (return). The results show that when setting an appropriate size of sample reviews, our approach receives a relatively stable precision, recall, and F-measure of requirements classification, in particular for non-functional requirements, which is meaningful and practical for APP developers to elicit requirements from user reviews.

Keywords—requirements identification; requirements classification; user review analysis

I. INTRODUCTION

Review function is provided by most APP distribution platforms (e.g., Apple App Store, Google Play) that allow users to rate and comment an APP after using it, which provides a feedback mechanism from users to developers and vendors of the APP. User reviews are recognized as a valuable source to improve APPs and increase the value for users [9][18], as the reviews help developers to better understand user needs as a type of collective knowledge [19]. However, existing APP platforms provide limited support for developers to systematically filter, aggregate, and classify user feedback to derive requirements [9]. User review and rating information has been investigated for technical and business purposes (e.g., APP price prediction) [11]. Pagano and Maalej collected the user reviews of the top 25 APPs from each of the 22 categories from App Store [1]. Based on the review data, they studied the content of user feedback and its impact on the user community. Chandy and Gu proposed an approach to automatically identify spam reviews in the iOS App Store [5]. However, there is little work on systematically and automatically identifying and

classifying requirements information from user reviews, which will significantly improve requirements elicitation and analysis in APP development. To this end, we propose an approach to automatically identify requirements information from user reviews and further classify them into functional (FR) and non-functional requirements (NFR), which are the basic classification of software requirements. For the practical application of the proposed approach, we further analyze the cost and return of our approach: how the size of sample reviews for keywords selection (i.e., the cost, described in Section III) affects the classification results in precision, recall, and F-measure (i.e., the return, presented in Section IV.C).

In the rest of this paper: Section II provides an overview of our proposed approach and the tool support. Section III describes the principles, TF-IDF technique, and process of selecting keywords for automated requirements identification and classification. Section IV presents the experiment material (user reviews of a popular APP iBooks) and the experiment results. The implications of the results are discussed in Section V. The threats to validity are described and analyzed in Section VII. Related work is discussed in Section VI. We conclude this work with further work directions in Section VIII.

II. APPROACH AND TOOL SUPPORT

When developing and continuously updating APPs, developers (especially requirements engineers) are responsible for being very much concerned about user experience and needs (e.g., privacy requirements [20]). If the requirements information from user reviews can be automatically identified and classified, it will significantly help developers and vendors to improve the quality and satisfaction of the APPs, for example, collecting critical and missing features for APP update. To this end, we propose an automated approach with tool support for identifying and classifying requirements from user reviews (see Fig. 1). There are two components in this tool: **User Reviews Extractor** is used to extract and collect user review information from APP platforms as raw data to be further processed, and **Requirements Identifier and Classifier** is used to identify and classify requirements from user reviews into FRs and NFRs.

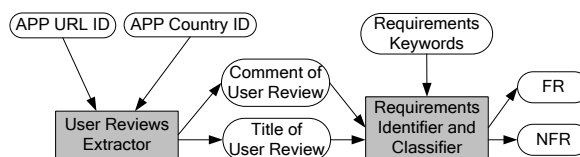


Figure 1. Proposed approach and tool architecture

* Corresponding author

This work is sponsored by the NSFC under Grant No. 61170025.

(DOI reference number: 10.18293/SEKE2015-063)

A. User Reviews Extractor

User Reviews Extractor uses *APP URL ID* and *APP Country ID* as input parameters to extract the user reviews of an APP from a specific APP platform. In the experiment of this work, we extracted and collected user reviews (including *comment* and *title* of user reviews) from APPs in Apple App Store. **User Reviews Extractor** uses the APIs provided by an open source package AppReviews¹ for accessing and retrieving the user reviews from App Store, which provides individual web portal in different countries with local languages. Each country store has its own *APP Country ID*, which allows us to access App Store for each country and retrieve the user review data of a specific APP using *APP URL ID*.

B. Requirements Identifier and Classifier

Requirements Identifier and Classifier is used to automatically identify requirements from user reviews and further classify them into FRs and NFRs. The inputs of Requirements Identifier and Classifier are the *title* and *comment* of user reviews and the extracted keywords (detailed in Section III) and the outputs are FRs and NFRs that are automatically classified. Note that some input user reviews may not contain any requirements information, which are namely spam reviews. These spam reviews² are roughly filtered out in **Phase 2** (i.e., pre-processing user reviews). The execution process of this component is composed of five sequential phases as shown in Fig. 2, which are further detailed in this section.

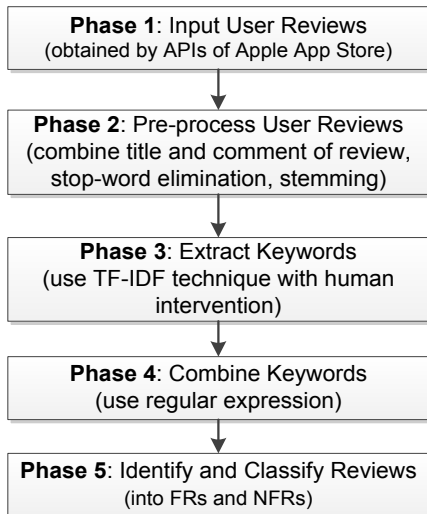


Figure 2. Processing phases of Requirements Identifier and Classifier

Phase 1: Input User Reviews to be processed: Preparing user reviews to be processed obtained by **User Reviews Extractor** as the input of **Requirements Identifier and Classifier**.

Phase 2: Pre-process User Reviews: User reviews obtained by **User Reviews Extractor** are pre-processed by automatically combining the *title* and *comment* of these

reviews as the target content, followed by eliminating punctuation marks (such as “,”, “.”) and stop words in natural language processing, like “a”, “the”, and “this”, filtering out spam reviews (e.g., the reviews less than three words), as well as word stemming [3].

Phase 3: Extract Keywords: In this phase, human experts (e.g., requirements engineers) first manually identify and classify a certain number of user reviews as NFRs or FRs, which are regarded as correct classifications, and then these classified NFRs and FRs are used as sample reviews to extract requirement keywords for automated identification and classification of NFRs and FRs respectively. These requirement keywords are automatically extracted from the sample reviews using TF-IDF technique [16] with human intervention by following the keywords extraction procedure detailed in Section III.B.

Phase 4: Combine Keywords: Requirements Identifier and Classifier combines the extracted keywords, the requirement keywords from each sample review (obtained from **Phase 3**), in various logical relationships (e.g., OR “[|]”) of regular expressions (e.g., *bug|crash*). These regular expressions are used to match (identify and classify) user requirements from user reviews in **Phase 5**. For example, for FR, *^is|are*choice\$*, which represents such phrases “is ... choice” or “are ... choices”.

Phase 5: Identify and Classify User Reviews: User requirements are identified and classified from the pre-processed content of reviews (obtained from **Phase 2**) using the regular expressions (obtained from **Phase 4**). A user review is automatically identified as requirement and classified into a NFR (or FR) using the regular expressions if the review can match the regular expressions (obtained from **Phase 3**). Note that, the identification and classification of requirements are performed in one step.

III. KEYWORDS SELECTION

A. Sample Reviews

According to the description in [14], a functional requirement specifies “a function that a system must be able to perform”, “what the product/system should do”, and a non-functional requirement is restricted to a set of specific qualities other than functionality: such as usability, reliability, and security. For example, a user review: “*the loss of the bookshelf look, the boring and ugly flat design plus the stark white background make it extremely difficult to read anything on this app.*” can be manually classified by domain experts as a NFR usability; another user review: “*at least give me the option of how I would prefer it to look.*” can be categorized as a FR that allows users to configure the style of UI. These manually identified and classified NFRs and FRs are used as sample reviews to extract keywords for NFR and FR identification and classification.

B. Keywords Extraction

As shown in Fig. 1, the requirement keywords are used to identify requirements information from user reviews and further to classify them into FRs and NFRs. The selection of

¹ <http://www.perculasoft.com/appreviews/>

² We are not intending to filter out all spam reviews, but only the obvious spams to improve the efficiency of subsequent processing.

the keywords is critical to the quality of the requirements identification and classification results.

In the field of information retrieval, Term Frequency - Inverse Document Frequency (TF-IDF) [16] is a statistic-based technique used to reflect how important a word is to a document in a collection or corpus. This technique has been successfully applied to text mining and classification (e.g., [15]). We use TF-IDF to calculate and evaluate the importance of a word extracted from each sample NFR (or FR) review to the set of sample NFR (or FR) reviews that are manually classified by domain experts. TF means the importance of a word extracted from each sample NFR (or FR) review to the sample review. The words that obtain a high TF-IDF score in each sample review require further checking by human experts, who judge and select the keywords which act as representative keywords of the sample review. For example, “*privacy*” is not considered as the keyword for FR, and “*feature*” is filtered out from the keywords for NFR. The selection criteria employed by human experts are very simple: for **FR**, the words which typically represent the NFR information should be excluded from the FR keywords (e.g., “*privacy*”, “*security*”, “*usability*”, and “*crash*”); for **NFR**, the words which typically represent the FR information should also be excluded from the NFR keywords (e.g., “*feature*” and “*choice*”).

The formulas for calculating the TF-IDF score of each word [16] are as follows (Formula (1) and (2) are used for calculating the TF-IDF score of NFR and FR words respectively), which are further explained below.

$$\text{Score}_{nfr}(w) = \frac{\text{freq}(Rv, w)}{|Rv|} \times \log \frac{|R(w)|}{Na(w)} \times \frac{Nnfr(w)}{Na(w)} \quad (1)$$

$$\text{Score}_{fr}(w) = \frac{\text{freq}(Rv, w)}{|Rv|} \times \log \frac{|R(w)|}{Na(w)} \times \frac{Nfr(w)}{Na(w)} \quad (2)$$

Each word w in a sample review (NFR or FR) will obtain a TF-IDF score $\text{Score}_{nfr}(w)$ or $\text{Score}_{fr}(w)$, which represents the importance of the word w in identifying and classifying user reviews. $\text{freq}(Rv, w)/|Rv|$ denotes the TF (term frequency) section of TF-IDF, in which $|Rv|$ refers to the quantity of all the words contained in the review Rv and $\text{freq}(Rv, w)$ represents the frequency of word w appearing in the sample review Rv . $\log(|R(w)|/Na(w))$ represents the IDF (inverse document frequency) section of TF-IDF, in which $Na(w)$ denotes the number of sample reviews that contain the word w , and $|R(w)|$ denotes the number of reviews to be classified that contain the word w . $Nnfr(w)$ or $Nfr(w)$ represents the number of NFR or FR sample reviews that contain the word w . $Nnfr(w)/Na(w)$ in Formula (1) or $Nfr(w)/Na(w)$ in Formula (2) implies if the word w is more densely distributed in the set of sample NFR or FR reviews, the word w is more important (i.e., $\text{Score}(w)$ is higher) in identifying and classifying NFRs or FRs from user reviews.

According to the obtained TF-IDF score of each word, the words are extracted from each sample review as representative requirement keywords of this sample review, and they are added into the requirement keywords set (duplicated keywords are removed). When keywords are extracted from all sample reviews and added to the keywords set, the keywords selection process is finished. The requirement keywords set is then used

to identify and classify requirements from user reviews. One user review can be classified as NFR or FR when it contains (can match) the requirement keywords of NFR or FR in the requirement keywords set.

IV. EXPERIMENT

A. Experiment Material

iBooks is a popular APP in the books category to read and buy books online through various Apple devices. This APP is provided for free in App Store. We decided to choose the user reviews of iBooks APP in English App Store as experiment material for the following reasons: (1) there are a large number of users of iBooks APP, which provide rich review data for the experiment; (2) the user reviews of this APP can be easily classified without the necessity of much domain knowledge, which improves the reliability of the experiment results (further discussed in Section VI); and (3) the review data in English is widely understandable which might act as benchmark data for other researchers to repeat this experiment using their own classification methods and tools.

B. Selected Keywords

As described in Section III, keywords are selected from a set of manually classified sample reviews. To investigate the cost and return of our approach, i.e., how the size of sample reviews (cost) for keywords selection affects the classification results (return), we provide increasing sizes of sample reviews as follows: 1, 3, 5, 7, 9, 10, 20, 30, 50, and 100. It is worth noting that these sets of sample reviews are independent of each other (i.e., one user review cannot exist in two sample sets). We then extracted the keywords from different size of sample reviews for identifying and classifying user requirements by following the keywords extraction procedure (in Section III.B). We first extracted the keywords from the latest “1” user review of iBooks APP, and then we iterated the keywords selection steps towards the remaining latest 3, 5, 7, 9, 10, 20, 30, 50, and 100 reviews from iBooks. Finally, all the selected keywords from each set of sample reviews are collected in an XML file and used to identify and classify requirements from the user reviews of iBooks. The requirement keywords, extracted using TF-IDF for each set of sample reviews, and further checked by human experts (see Section III.B), are available online³.

C. Experiment Results

To investigate the effectiveness of our approach, we compare the manual classification results by experts (the two authors), which act as ground truth, with the identification and classification results. The experiment use 1000 user reviews as experiment material retrieved from iBooks APP in English App Store. For the practical application of the approach, we further analyze the cost and return of our approach as discussed in Section IV.B, i.e., the experiment results are further evaluated and compared with different sizes of sample reviews (cost) using precision, recall, and F-measure of the classification results (return). The experiment results are presented below.

³ <http://www.cs.rug.nl/search/uploads/Resources/TF-IDF-Keywords.zip>

In iBooks APP from English App Store, we obtained 217 (set **B** in Fig. 3) user reviews containing FR information and 622 user reviews containing NFR information among the 1000 user reviews (some user reviews may contain both FR and NFR information) by manual classification (i.e., the ground truth). To examine that how the size of sample reviews affects the classification results, we provide the sizes of sample reviews as follows: 1, 3, 5, 7, 9, 10, 20, 30, 50, and 100, which is shown in the x-axis of Fig. 4 and Fig. 5.

We evaluate our approach through comparing automated classification results with manual classification results. We use F-measure which is a combination of precision and recall used in the evaluation of information retrieval systems [2] to measure the overall performance of the automated classification results.

In this work, precision means the percentage of user reviews that are correctly classified as FRs or NFRs compared to all the classified results (i.e., set **A** divided by **C** as illustrated in Fig. 3), and the recall refers to the percentage of user reviews that are correctly classified as FRs or NFRs compared to the manual classification results - the ground truth (i.e., set **A** divided by **B** in Fig. 3).

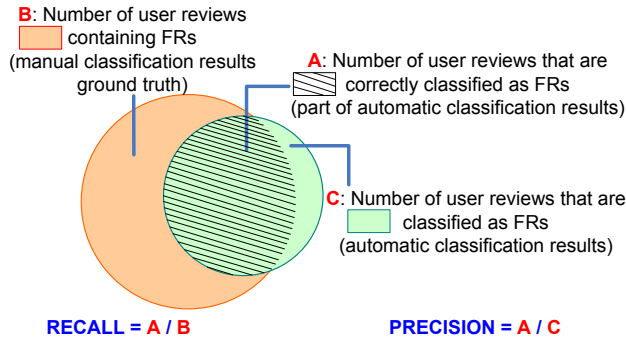


Figure 3. Recall and Precision calculation for classification results evaluation

We calculate and get the evaluation results of FR and NFR classification as shown in TABLE I and TABLE II respectively (including the size of sample reviews, the number of extracted keywords using TF-IDF, Precision, Recall, and F-measure for FR and NFR classification). Fig. 4 and Fig. 5 show the trend line of Recall, Precision, and F-measure for FR and NFR classification results on iBooks user reviews along with different sample sizes of user reviews. These two figures show that the value of F-measure (represented in blue line) is significantly increased as the size (number) of sample reviews increases, but when the size of sample reviews reaches a certain threshold, the value of F-measure tends to be stable. The possible explanation of the experiment results and their implications will be discussed in Section V.

TABLE I. RESULTS OF AUTOMATED FR CLASSIFICATION ON 1000 iBOOKS USER REVIEWS WITH DIFFERENT SIZES OF SAMPLE REVIEWS

Sample Size	No. of Keywords	Precision	Recall	F-measure
1	5	0.000	0.000	0.000
3	15	0.406	0.129	0.196
5	18	0.454	0.184	0.262
7	20	0.469	0.207	0.288

9	24	0.404	0.281	0.332
10	26	0.394	0.249	0.305
20	53	0.385	0.525	0.444
30	56	0.356	0.710	0.474
50	75	0.383	0.636	0.478
100	116	0.350	0.760	0.479

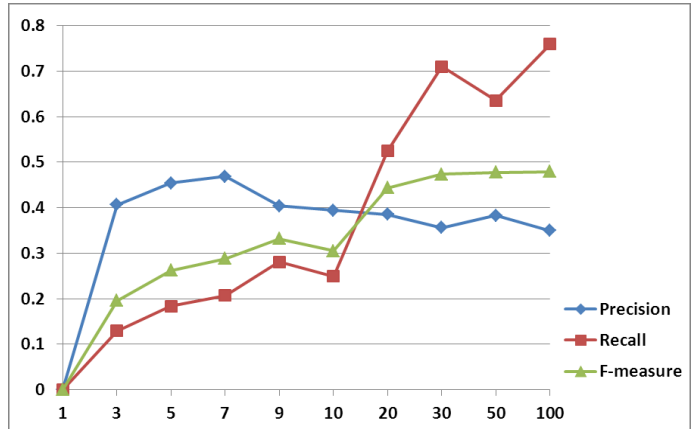


Figure 4. Trend lines of Precision, Recall, & F-measure for FR classification on 1000 iBooks user reviews with different sizes of sample reviews

TABLE II. RESULTS OF AUTOMATED NFR CLASSIFICATION ON 1000 iBOOKS USER REVIEWS WITH DIFFERENT SIZES OF SAMPLE REVIEWS

Sample Size	No. of Keywords	Precision	Recall	F-measure
1	5	0.909	0.032	0.062
3	12	0.837	0.215	0.342
5	16	0.836	0.418	0.557
7	25	0.816	0.740	0.776
9	22	0.820	0.698	0.754
10	28	0.826	0.704	0.760
20	43	0.795	0.810	0.803
30	57	0.758	0.897	0.823
50	82	0.761	0.895	0.823
100	104	0.745	0.924	0.825

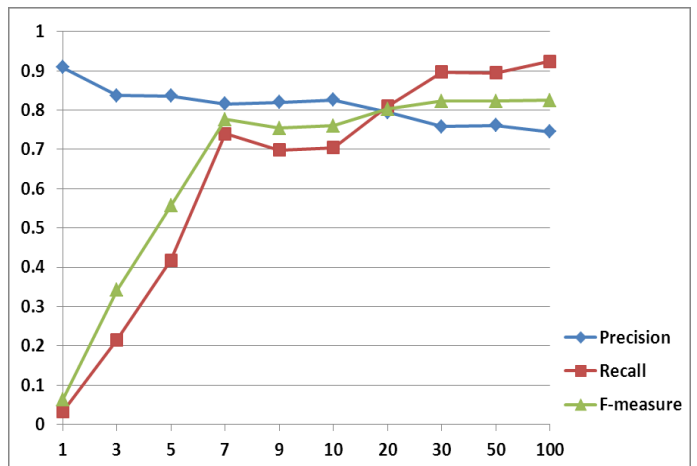


Figure 5. Trend lines of Recall, Precision, & F-measure for NFR classification on 1000 iBooks user reviews with different sizes of sample reviews

V. DISCUSSION

We explain the experiment results and discuss their implications according to the visualization in Fig. 4 and Fig. 5.

1) In both Fig. 4 and Fig. 5, the value of F-measure dramatically increases when the sample size increases initially (e.g., from 1 to 20 for FR classification, and from 1 to 7 for NFR classification), and after that size (number) the value of F-measure tends to be stable. This result implies that there is a certain threshold of sample size that can achieve a comparably good and balanced classification results without the necessity to increase the sample size unceasingly.

2) The significant difference between the thresholds of sample size for FR and NFR classification (20 vs. 7 in this experiment) implies that NFR classification requires less sample reviews to get a decent set of keywords reaching a stable F-measure than FR classification, which is reasonable since the FR keywords are more domain-dependent than the NFR keywords. This may also explain a relatively higher F-measure (e.g., when the sample size is 100) of the NFR classification results (0.825) than the FR results (0.479).

3) From both Fig. 4 and Fig. 5, it can be found that the three trend lines of Precision, Recall, and F-measure have an intersection point (e.g., a sample size (number) between 10 to 20 for FR classification, and 20 for NFR classification), and after that size (number) the value of F-measure tends to be stable. This intersection point seems providing a reliable way to decide the threshold of sample size as discussed in point (1) for a balanced (cost vs. return) classification results. But this conjecture should be validated with more experiments (APPs).

VI. THREATS TO VALIDITY

We discuss the threats to the validity by following the guidelines in [4] and how they are partially mitigated.

Construct validity: We use F-measure from information retrieval theory to evaluate the requirements classification results. The automated requirements classification with our approach is basically an information retrieval activity since both of them use keywords to get results.

Internal validity focuses on the unknown factors that may have an influence on the study results. This experiment is a study about the performance of the proposed approach (to what extent the approach can identify and classify requirements from user reviews) using descriptive statistics. In other words, we did not investigate and intend to establish any causal relationship between the identification and classification results and the factors that may impact the results in this study, and the threats to internal validity are minimal.

External validity: We applied our approach to a popular APP from the books category (application domain) in English App Store with promising results. This experiment can be repeated with APPs in other domains and languages to improve the applicability and generalizability of the proposed approach.

Reliability: The manual requirements classification results by experts are regarded as ground truth to be compared with the automated classification results for the evaluation (in

Section IV.C), but the manual classification results might be different when conducted by different experts, which makes the evaluation results not reliable. We tried to mitigate the influence of this issue by three measures: (1) we selected a general APP iBooks and its reviews can be reliably classified without the need of much domain knowledge. (2) the manual classification results by the first author were checked by the second author, and any disagreement on the classification results was discussed and resolved. (3) the manual classification results were further examined by 10 master students, who major in software engineering through voting. We also set criteria (see Section III.B) to select requirement keywords by experts, and this mitigates the bias when different experts select representative keywords from the keywords obtained by TF-IDF.

VII. RELATED WORK

We summarize and discuss relevant work and their relationship to our work in this section.

Chen and his colleagues proposed a method to automatically mine informative reviews for APP developers, and further rank these informative reviews [21]. Our work aims to identify and classify the informative reviews that contain requirement information as functional and non-functional requirements.

Khalid and his colleagues [17] focused on low star-rating user reviews of free iOS APPs, and identified 12 types of complaints that users complain about. They found that functional errors, feature requests, and APP crashes are the most frequent complaints, which supports that user reviews are indeed rich source of requirements.

Pagano and Maalej presented an empirical study on user feedback in the App Store [1]. They mainly discussed the usage of user feedback by the users, the content of user feedback, and its impact on the user community in the App Store, through analyzing the App Store review data with statistical approaches. They also discussed the impact of user feedback to requirements engineering, which inspires our work.

Galvis Carreño and Winbladh focused on changing requirements and creating new requirements using the topics identified from user reviews [13], while our work is different in that we try to identify original requirements and classify them from user reviews. The outcome of our approach can act as input (identified and classified user requirements) of [13] for topics identification in requirements evolution.

Chandy and Gu proposed an approach to automatically identify spam reviews in the iOS App Store [5], which compared the performance of a baseline Decision Tree model with a novel Latent Class graphical model to the classification results of App review spam. The difference of their work to our work is that they employ data mining techniques and focus on spam identification.

Finkelstein and his colleagues [11] introduced a method to extract feature and price information of the APPs in the Blackberry App Store for an analysis that combines technical, business, and customer properties. The analysis results are further used as the input to predict the prices of APPs with

case-based reasoning, while our work focuses on the extraction of user requirements information from APP user reviews.

Sagar and Abirami investigated conceptual modeling of FRs in natural language [10]. For the purpose of visualizing the FRs, they focus on automated extraction of concepts and their relationships to create a conceptual model based on linguistic aspects of the English language. Their work could be useful to develop the conceptual model for FR identification and classification from user reviews.

The work on mining general and APP repositories focuses on analyzing the feature information among user reviews, and understanding their inter-relationships with other factors, e.g., rating, price, downloads, and code [6][7][11]. Our approach tries to combine App Store reviews mining and requirements engineering to help developers understand the trend of software products in order to improve their APPs.

VIII. CONCLUSIONS AND FUTRUE WORK

In this paper, we present an approach, which can automatically identify and classify requirements from user reviews. We validated the proposed approach with user reviews collected from a popular APP iBooks in English App Store, and further investigated the cost and return of our approach: how the size of sample reviews for keywords selection (cost) affects the classification results in precision, recall, and F-measure (return). The results show that when setting an appropriate size of sample reviews, our approach receives a relatively stable precision, recall, and F-measure of requirements classification, in particular for non-functional requirements, which is meaningful and practical for APP developers to elicit requirements from user reviews. In the next step, the approach can be improved in three promising aspects:

1) To validate our approach with user reviews of APPs from other application domains (e.g., social networking, finance) and languages (e.g., East Asian languages) and perform comparative studies with other identification and classification approaches (e.g., through data mining, machine learning techniques) in order to mitigate the threats to the external validity of the results.

2) The identified and classified requirements can be further prioritized to show their importance when hundreds-and-thousands of requirements flooding to developers [8]. The potential factors for prioritizing requirements from user reviews are different from those for general requirements prioritization, for example, rating information, length of user review, and stickiness or importance of the user who submitted the review. All these factors are expected to have an influence on prioritizing requirements from use reviews, and other potential factors should also be considered depending on the needs of requirements prioritization in context.

3) Functional and non-functional requirements are not independent of each other [14], for example, one NFR may impact many FRs, which is an important part of requirements traceability. The potential relationships between classified FRs and NFRs can be promisingly identified through their source analysis, e.g., the user-review relationships.

REFERENCES

- [1] D. Pagano and W. Maalej, "User feedback in the AppStore: An empirical study," In: Proceedings of the 21st International Conference on Requirements Engineering (RE), IEEE, 2013, pp. 125-134.
- [2] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. vol. 463. New York. ACM Press, 1999.
- [3] J. B. Lovins, Development of A Stemming Algorithm. MIT Information Processing Group, Electronic Systems Laboratory, 1968.
- [4] F. Shull, J. Singer, and D. I. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering. Springer, 2008.
- [5] R. Chandy and Gu. H, "Identifying spam in the iOS app store," In: Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality (WebQuality), ACM, 2012, pp. 56-59.
- [6] M. Harman, Y. Jia, and Y. Zhang, "App Store mining and analysis: MSR for App Stores," In: Proceedings of the 9th Working Conference on Mining Software Repositories (MSR), IEEE, 2012, pp. 108-111.
- [7] A. Zaidman, B. Van Rompaey, S. Demeyer, and A. van Deursen, "Mining software repositories to study co-evolution of production & test code". In: Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST), IEEE, 2008, pp. 220-229.
- [8] S. Gartner and K. Schneider, "A method for prioritizing end-user feedback for requirements engineering," In: Proceedings of the 5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE, 2012, pp. 47-49.
- [9] U. Abelein, H. Sharp, and B. Paech, "Does involving users in software development really influence system success?" IEEE Software, IEEE, 30(6):17-23, 2013.
- [10] V. Sagar and S. Abirami, "Conceptual modeling of natural language functional requirements," Journal of Systems and Software, Elsevier, 88(2):25-41, 2014.
- [11] A. Finkelstein, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "Mining App Stores: Extracting technical, business and customer rating information for analysis and prediction," Research Note, RN/13/21, 2013.
- [12] J. Cleland-Huang, R. Settini, X. Zou, and P. Solc, "Automated classification of non-functional requirements," Requirements Engineering, Springer, 12(2):103-120, 2007.
- [13] L.V. Galvis Carreño, and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," In: Proceedings of the 35th International Conference on Software Engineering (ICSE). IEEE, 2013, pp. 582-591.
- [14] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, Non-functional Requirements in Software Engineering. Springer, 2000.
- [15] M.K. Dalal, and M.A. Zaveri, "Automatic text classification of sports blog data", In: Proceedings of the 2012 Computing, Communications and Applications Conference (ComComAp), IEEE, pp.219-222, 2012.
- [16] J. Ramos, "Using tf-idf to determine word relevance in document queries," In: Proceedings of the 1st Instructional Conference on Machine Learning (iCML), 2003, pp. 119-122.
- [17] H. Khalid, E. Shihab, M. Nagappan, and A. Hassan, "What do mobile App users complain about? A study on free iOS Apps," IEEE Software, IEEE, DOI: <http://dx.doi.org/10.1109/MS.2014.50>, 2014.
- [18] M. Bano and D. Zowghi, "User involvement in software development and system success: A systematic literature review," In: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (ESEM), ACM, 2013, pp. 125-130.
- [19] P. Liang, P. Avgeriou, K. He, and L. Xu, "From collective knowledge to intelligence: pre-requirements analysis of large and complex systems," In: Proceedings of the 1st Workshop on Web 2.0 for Software Engineering (Web2SE), ACM, 2010, pp. 26-30.
- [20] K. Thomas, A.K. Bandara, B.A. Price, and B. Nuseibeh, "Distilling privacy requirements for mobile applications," In: Proceedings of the 36th International Conference on Software Engineering (ICSE), ACM, 2014, pp. 871-882.
- [21] N. Chen, J. Lin, S.C.H. Hoi, X. Xiao, and B. Zhang, "AR-Miner: mining informative reviews for developers from mobile app marketplace," In: Proceedings of the 36th International Conference on Software Engineering (ICSE), ACM, 2014, pp. 767-778.

MoLVERIC: An Inspection Technique for MoLIC Diagrams

Adriana Lopes, Anna Marques and Tayana Conte
USES Research Group
Instituto de Computação, Universidade Federal do
Amazonas (UFAM)
Manaus, AM - Brazil
{adriana,anna.beatriz,tayana}@icomp.ufam.edu.br

Simone Diniz Junqueira Barbosa
Semiotic Engineering Research Group
Departamento de Informática, PUC-Rio
Rio de Janeiro, RJ - Brazil
simone@inf.puc-rio.br

Abstract— During interaction design, interaction models are developed to help design adequate user interaction with the system. MoLIC (Modeling Language for Interaction as Conversation) is a language used to represent an interaction model, which can then be used as a basis for building other artifacts, such as mockups. However, inspections are necessary to verify whether the MoLIC diagrams are complete, consistent, unambiguous, and contain few or no defects, to avoid propagating preventable defects to derived artifacts. In this paper, we present MoLVERIC, a technique for the inspection of MoLIC diagrams that uses cards with verification items and employs principles of gamification. Furthermore, we discuss the results of a pilot study conducted to analyze the feasibility of this technique.

Keywords— component; Interaction Design; Interaction Modeling; Verification; Inspection Technique.

I. INTRODUCTION

Interaction design aims to design systems that are easy to learn, effective when used and capable of providing a rewarding experience to the user [1]. In this context, Semiotic Engineering [2], a theory of Human-Computer Interaction, deals with interaction as a communication process between the user and the system, through its user interface. Based on Semiotic Engineering, Barbosa and Paula [3] proposed MoLIC (acronym for Modeling Language for Interaction as Conversation), a language to model this interaction. MoLIC diagrams can be used by different practitioners involved in the development of systems for modeling a global view of the application's apparent behavior. Moreover, MoLIC diagrams can serve as a basis for the construction of other artifacts in the development of interactive systems, such as mockups. Santana et al. [4] proposed the use of a communication theory called Grice's Cooperative Principle [5] for inspection of MoLIC diagrams with the focus on user communication.

However, MoLIC diagrams should be also verified with respect to their consistency, completeness and comprehensibility in order to reduce the number of defects and to prevent them from propagating to derived artifacts. To investigate the quality of MoLIC diagrams, we conducted a preliminary study that has identified several defects which had been inserted during interaction modeling. Through such inspection, the propagation of defects in MoLIC diagrams can

be avoided, reducing the cost of correcting such defects in later stages of the software development process [6].

In this paper we present MoLVERIC, a technique for inspecting MoLIC interaction diagrams. The MoLVERIC technique was developed based on the defects that were found in a preliminary study. The purpose of MoLVERIC is to provide a simple way to identify defects in MoLIC diagrams, so that the technique can be easily adopted by both academy and industry. With this technique, we intend to prevent possible defects from being transferred to artifacts that are constructed based on the MoLIC diagrams. To assess whether MoLVERIC can support inspectors in detecting defects, we conducted a pilot study, whose results have provided evidence of the feasibility of MoLVERIC to inspect MoLIC diagrams.

The remainder of this paper is organized as follows. Section II presents the MoLIC language. Section III describes the defect types found in MoLIC diagrams in our preliminary study. Then, Section IV presents the MoLVERIC technique, and Section V describes the pilot study conducted to evaluate MoLVERIC. Finally, we present some concluding remarks and discuss future work.

II. MoLIC

MoLIC is based on Semiotic Engineering [2], a theory of HCI with particular focus on the communication between the designer and the user mediated by interactive systems, through the designer's deputy, which is the user interface. The designer's deputy "talks" with the user, enabling the mediated designer-to-user communication about the designer's view and design decisions. Because the designer-to-user message is about the messages users can exchange with the user interface, it is called a *metacommunication message*. MoLIC was devised to represent this message.

A MoLIC diagram can be created after the requirements elicitation, within the analysis stage of the software development process. The purpose is to promote the designers' and developers' reflection on the interaction alternatives that they intend to provide to the users [7]. To illustrate the MoLIC diagrammatic notation [7], Figure 1 represents a diagram of a simple system for calculating a person's Body Mass Index (BMI). The basic elements of a MoLIC diagram are the following:

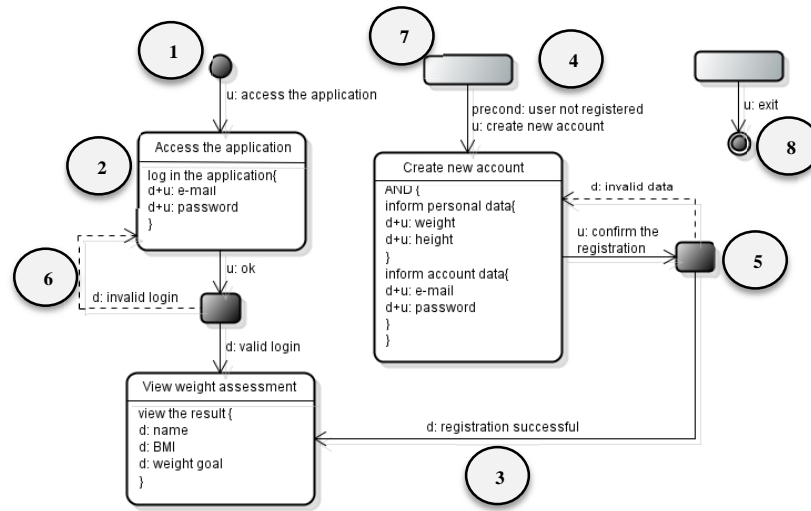


Figure 1. A MoLIC diagrams example.

1. *Opening point*: Indicates where the interaction can start, *i.e.*, when the user accesses the system. It is represented by a filled black circle.
2. *Scene*: Represented in the diagram as a rounded rectangle. The scene shows the moment in the interaction where the user decides how the conversation should proceed. The top compartment contains the topic of the scene and represents the user's goal. The second compartment details the following elements:
 - a) *Signs*: represent the information involved in the utterances issued by the user (*i.e.*, user input) and by the designer's deputy (*i.e.*, system output) during the dialogues. In Figure 1, we have the following signs in the "Access the application" scene: "e-mail and password".
 - b) *Utterances*: constitute the dialogue and specify who is emitting the sign: whether it is the user (u) or the designer's deputy (d). The signs issued only by the designer's deputy (e.g. system output) are preceded by "d". In Figure 1, we have the following signs and utterances in the "View weight assessment" scene, e.g. "d: name, d: BMI and d: weight goal", all emitted by the designer's deputy, because they are just for providing information to the user. When the designer allows the user to talk about the signs, e.g., when it involves user input, we say that both the designer and the user emit the sign, which is then preceded by "d+u". In Figure 1, we have such signs in the "Access the application" scene: "d+u: e-mail, d+u: password", for example.
 - c) *Dialogues*: compose a conversation about a topic, and consist of utterances on signs. In Figure 1, one example of dialogue is "view the result" (in "View weight assessment").
 - d) *Structures of dialogues*: in some cases, the dialogues can be composed by other dialogues according to some structure. In these cases, these structures can be represented by the reserved words SEQ, XOR, OR or AND. The SEQ structure represents the dialogues that must be exchanged in the specified sequence. The XOR

structure represents mutually exclusive dialogues. The structure OR represents the choice of exchanging one or more dialogues. The structure AND represents the use of all dialogs, but not in a predefined sequence. In Figure 1, the AND structure represents the use of all dialogs "inform personal data" and "inform account data".

3. *Transition Utterance*: Represents turn-taking, or rather turn-giving, where either the user or the designer's deputy gives the turn to the other, for instance, to change the topic of the conversation, as described below:
 - a) *User Utterance*: represents the user's intent to proceed with the conversation in a given direction. It is represented by an arrow in the diagram, labeled with a user utterance indicator (u:), e.g. "u: ok" in Figure 1.
 - b) *Designer Utterance*: represents the designer's deputy's answer to a user utterance, typically provided after a system process. It is represented by an arrow in the diagram, labeled with a designer utterance indicator (d:) e.g. "d: valid login" in the Figure 1.
4. *Precond*: represents a necessary precondition in the diagram. In Figure 1, the user can only create an account if (s)he is not yet registered. This precondition is represented before the user utterance through "precond: user not registered".
5. *System process*: It is represented through a black box in the diagram. It represents the internal processing (of a user request) which needs to provide adequate feedback to the user, *i.e.*, when there are different outcomes possible.
6. *Breakdown recovery utterance*: is a type of utterance provided to help the user recover from a communication breakdown. It is represented by a dashed directed line in the diagram with the corresponding utterance, e.g. "d: invalid data" in Figure 1.
7. *Ubiquitous access*: represents an opportunity for the user to change the topic of the conversation from any other scene, to achieve an objective different from the current one. It is represented through a gray rounded rectangle.

8. *Closing point*: represents the end of the interaction, when the user leaves the system. It is represented as a filled black circle within a circle with no padding.

III. TYPES OF DEFECTS IN MOLIC DIAGRAMS

The types of defects that can be found in a MoLIC diagram were defined based on the taxonomy presented by Travassos et al. [6], as shown in Table I. Using a taxonomy of defects is important in order to assist the inspectors (practitioners who carry out the inspection) in the identification and categorization of defects.

TABLE I. DEFECTS TAXONOMY FOR MOLIC DIAGRAMS

Types of Defects	Description of Defects
Omission	Omission or negligence of any information necessary to solve the problem in the interaction diagram.
Ambiguity	Unclear definition of a certain information in the interaction diagram, which may lead to multiple interpretations.
Incorrect Fact	Misuse of the interaction diagram elements.
Inconsistency	Conflicting information between the interaction diagram elements and the information needed to solve the problem.
Extraneous Information	Unnecessary information included in the interaction diagram (i.e., information that is not needed to solve the problem).

We conducted a preliminary study with 13 subjects in order to evaluate the quality of some sample MoLIC diagrams. Study subjects were undergraduate students (in their final year) and graduate students in a Computer Science course. Before the study, all subjects received training in using the MoLIC language to model the interaction of a system, because the subjects had no experience with MoLIC diagrams. Each subject individually built the MoLIC diagram, using computers with a MoLIC designer tool¹ installed.

After the study execution, two experts in MoLIC diagrams verified the produced diagrams. These experts found the defects in the artifacts and categorized them according to the taxonomy shown in Table I. During the analysis, the repeated defects were informed. In the total we identified: 13 omissions, 5 extraneous informations, 1 ambiguity, 7 incorrect fact and 5 inconsistencies.

TABLE II. DEFECTS FOUND IN MOLIC DIAGRAMS FOR EACH TYPE OF DEFECT

Type of Defect	Subjects (S)	Example of Defect Found in the MoLIC Diagram
Omission	S2, S4, S5, S12	Did not use the notation of the utterance for the user (u:) and the designer's deputy (d:).
Ambiguity	S12	Used two user transition utterances for the same goal, thus providing multiple interpretations for the user request.
Incorrect Fact	S4, S8, S9, S11	Used verbs that do not represent the user goals.
Inconsistency	S1, S5, S6, S11,	Used a transition arrow direction inconsistent with the sequence of interaction scenes.
Extraneous Information	S4, S8,	Represented some scenes that were not in the context of the interaction scenario.

In Table II, we present one detailed example of each type of defect found in MoLIC diagrams. In addition, the defects are associated with the subjects who developed the diagrams containing the defects. As noted in this study, MoLIC diagrams

may contain defects, which can impair the understanding of the practitioners involved during the development of the systems. Therefore, it is important to perform inspections in MoLIC diagrams before they are used in the next phases of the systems development, for creating new artifacts such as mockups, for instance.

IV. MOLVERIC

Inspection of system artifacts during development has been shown to improve the quality of the system and reduce development costs. An inspection is a method for identifying defects in early stages of the development [6]. Conducting inspections is essential because design defects can directly affect the quality of the systems [6]. One of the most widely used methods in the inspection is Checklist [8]. Checklists provide support for inspectors during the defects detection through verification items [8].

MoLVERIC is a checklist-based inspection technique, developed with the goal of assisting practitioners in the inspection of MoLIC diagrams. All verification items of MoLVERIC were developed based on the defects found in the preliminary study, as described in Section III. The verification items assess both the consistency of MoLIC diagrams with the interaction scenario/system requirements; and the notation used in the MoLIC diagrams. To motivate the inspection of MoLIC diagrams, MoLVERIC employs gamification [10] techniques. Each card corresponds to a verification item and includes the number of points awarded to the inspector each time he/she finds the defect described in the card. Each verification item assists the inspector in reporting the type of the identified defect. With respect to the score, items that verify defects that compromise the purpose or understanding of the diagram award 20 points, whereas items that verify a syntax defect, which does not compromise the objective or understanding of the diagram, award 10 points. The verification cards are divided into categories corresponding to the elements of the MoLIC diagram, such as Scene, Transition Utterance and Signs. During the development of the verification items for each category of the MoLIC diagram elements, we noted that some defects were related to more than one element. Therefore, we developed different verification items for single elements and for related elements. To do so, we developed two types of cards: Regular Cards and High Cards.

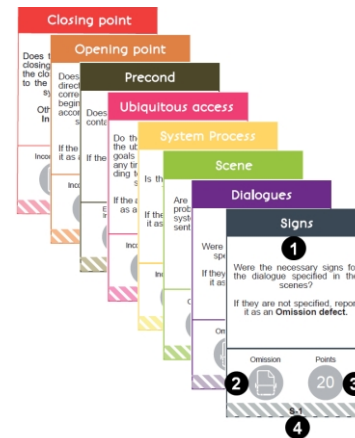


Figure 2: Verification items from MoLVERIC: Regular Cards

1. <https://code.google.com/p/mollic-designer/>

Regular Cards have one-to-one correspondences to elements of the MoLIC diagram. There are Regular Cards for the following elements: Scene, System Process, Opening Point, Closing Point, Ubiquitous Access, Precond, Dialogues and Signs. In Figure 2, the Regular Cards are presented using the following structure: (1) Description of the verification item, to assist the inspector in the identification of defects; (2) Type of defect to be reported; (3) Points of the card and (4) Code of the verification item.

High Cards have verification items for elements related to other elements in the MoLIC diagram. The goal of the High Cards is reduce the inspection time for elements that are related in the diagram. There are High Cards for the elements Transition Utterance and Breakdown Recovery Utterance. A Transition Utterance element is related to the elements: Scene, System Process, Opening Point, Closing Point and Ubiquitous Access. Figure 3 shows an example of an inspection using a High Card for the Transition Utterance element, where the following verification item is used for the related elements: “Do the utterances show who uttered them (“u:” for the user and “d:” for the designer’s deputy)? If the answer is negative, report it as an Omission defect.”

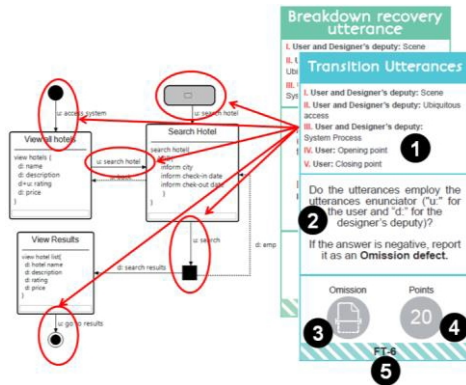


Figure 3: Verification items from MoLVERIC: High Cards.

A Breakdown Recovery Utterance is related with the elements: Scene, System Process and Ubiquitous Access. The High Cards have the following structure in each card, as shown in Figure 4: (1) Elements in the MoLIC diagram related with the elements of the card; (2) Description of the verification item, to assist the inspector in the identification of defects; (3) Type of defect to be reported; (4) Points of the card and (5) Code of the verification item. Furthermore, there are instructions for the inspectors in order to show how to use MoLVERIC.

V. PILOT STUDY OF MOLVERIC

In this pilot study, we did not use other inspection technique for MoLIC diagrams to compare with the MoLVERIC. The reason for this is that the only other known technique (Grice's Cooperative Principle) has a different focus. The pilot study activities are described as follows.

A. Pilot Study Planning

During the planning stage, we defined the resources needed for implementing the study. Therefore, we planned the execution environment, as well as the artifacts, as follows:

- *Environment:* The study was conducted in an academic environment, where new technologies are tested before being transferred to industry [11].
- *Artifacts:* (i) The MoLIC diagrams built in the preliminary study described in Section III; (ii) forms for the subjects to report the identified defects; (iii) instructions for using the technique; (iv) post-study questionnaire to be answered by each subject, to collect their opinions about the technique.
- *Subjects:* For the study, two subjects who had developed MoLIC diagrams in previous projects were chosen to inspect the diagrams. The subjects were graduate students in Computer Science.
- *Training:* The subjects received training on the types of defects and on the use of MoLVERIC.

B. Pilot Study Execution

During the study, each subject executed the inspection individually. After the study, we analyzed the defects reported in the forms and the post-study questionnaires.

C. Results Analysis

After the execution of the pilot study, we verified whether the technique achieved the goal of detecting defects. The oracle of defects contained a total of 24 defects (some defects are repeated in the elements signs, scenes and transition utterance). The number of defects, the inspection time and the indicators of effectiveness and efficiency of each subject are described in Table III. The effectiveness was calculated using the number of defects found by the subjects divided by the total number of defects from the oracle. The efficiency was calculated on the number of defects found divided by the time of inspection of each subject.

TABLE III. RESULTS PER SUBJECTS

Subjects	Number Defects	False Positive	Time Hours	Effectiveness	Efficiency
S1	17	1	1.61	70.83%	10.55
S2	16	3	1.15	66.66%	13.91

Analyzing the effectiveness indicator, we can see that the inspectors were able to identify more than 66% of the defects. This is a good result in terms of effectiveness when compared to the indicators achieved by other inspection techniques [12] and, as such, it indicates the feasibility of MoLVERIC. However, it is still necessary to perform a controlled experiment to compare the effectiveness of MoLVERIC with other techniques for identifying defects in interaction models. Regarding efficiency, the subjects found 10.55 and 13.91 defects per hour. However, as the number of defects is directly dependent on the inspected models, is not suitable to compare the results of efficiency from this pilot study with the results of other techniques.

To understand the opinions of the subjects, the answers to the post-questionnaire were analyzed. Regarding the ease of use of the technique, the subjects indicated the following:

“MoLVERIC helps to remember the elements that I should verify and the types of defects I should inspect, for example:

Omission, Incorrect Fact, Inconsistency, Extraneous Information and Ambiguity.” (S1)

“The technique provides an inspection guide. This guide does not leave the inspector lost. The technique makes the inspection process fun.” (S2)

However, subjects also reported negative aspects regarding the ease of use of the technique:

“I think that it takes a long time to learn to use the technique.” (S1)

“I had trouble remembering some terms.” (S2)

Regarding the quote from S1 about the negative aspects of the technique, this can be related to the amount of categories and the related items. However, due to the small number of subjects in this pilot study, this result cannot be considered conclusive. This aspect will be examined in future research with MoLVERIC. To understand how the subjects use MoLVERIC during the inspection, they answered the following question in the post-questionnaire: “Is the structure of MoLVERIC suitable for the inspection of the MoLIC diagrams in the way you inspect an artifact?”

“I think this is a good way to inspect. The structure of the cards is good.” (S1)

“Yes, it directs the structure according to the MoLIC diagram.” (S2)

However, during the study, the two subjects had difficulties in understanding the Precond category, an element used to specify a necessary precondition together with the Transition Utterance element. Regarding the other categories during the study, the subjects had no difficulty in the use of the other cards. During the analysis of the defects reported by the subjects, we verified that the subjects had no problems in understanding each inspected element, i.e., they reported them correctly according to the code for each card. However, observing the quotation from S2 about the negative aspects, there is a certain difficulty in understanding the term “issuer” of the signs, which refers to the “d” for the designer and “u” for the user. Furthermore, subjects responded positively to the question “Would you recommend this technique for designers who use the interaction modeling with MoLIC?” Both inspectors indicated that they would recommend the use of MoLVERIC. Based on the analysis of the results of this pilot study, it was possible to obtain indicators of the feasibility of using the technique.

VI. CONCLUDING REMARKS AND FUTURE WORK

The purpose of this paper was to present the results of the pilot study to evaluate the feasibility of MoLVERIC. The analysis study allowed us to identify problems during the use of the technique, as well as terms which were not clear and verification items that were not appropriate. Based on these results, we are improving MoLVERIC, making the verification items clearer. The results of the pilot study provided evidence that MoLVERIC assists in detecting defects. However, these results cannot be considered conclusive, and it is necessary to carry out a controlled experiment with a larger quantity of subjects. Analyzing the perception of the subjects on

recommending MoLVERIC, there was mostly positive feedback from the subjects. This may be an indication that the technique is suitable for the inspectors of MoLIC diagrams. The results of the pilot study provided initial evidence to the feasibility of MoLVERIC to inspect MoLIC diagrams.

As future work, we intend to carry out a controlled study with MoLVERIC to reinforce the results obtained in the study pilot. In this next study, we expect to evaluate more precisely the effectiveness and efficiency of MoLVERIC, so that it can be adopted by the industry and academy in the future. Furthermore, we intend to conduct an empirical study to analyze the quality of artifacts developed from the MoLIC diagrams, after the inspection with the MoLVERIC.

ACKNOWLEDGMENT

We thank the two graduate students for their participation in the experiment. We would like to acknowledge the financial support granted by FAPEAM (Foundation for Research Support of the Amazonas State) through processes numbers: 062.00146/2012; 062.00600/2014; 062.00578/2014; 01135/2011 and PAPE 004/2015; and CNPq processes 308490/2012-6, 453996/2014-0, and 460627/2014-7.

REFERENCES

- [1] Y. Rogers, H. Sharp, J. Preece. Interaction Design: Beyond Human-Computer Interaction, 4th Edition. John Wiley & Sons, 2015.
- [2] C. S. De Souza, The Semiotic Engineering of Human-Computer Interaction (Acting with Technology). The MIT Press, 2005.
- [3] S. D. J. Barbosa, M. G. Paula, Designing and Evaluating Interaction as Conversation: a Modeling Language based on Semiotic Engineering. In Interactive Systems. Design, Specification, and Verification. 10th DSV-IS Workshop, pp. 16–33, 2003.
- [4] B.S. Silva, V.C.O. Aureliano, S.D.J. Barbosa. Extreme designing: binding sketching to an interaction model in a streamlined HCI design approach. In Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais, pp. 101 – 109, 2006.
- [5] B. L. Davies. Grice’s cooperative principle: meaning and rationality. Journal of Pragmatics 39, 2308–2331, 2007.
- [6] G. H. Travassos, F. Shull, J. Carver, Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language. Advances in Computer, vol. 54, pp. 35 – 98, 2001.
- [7] A. Lopes, A. B. Marques, S. D. J. Barbosa and T. Conte. Evaluating HCI Design with Interaction Modeling and Mockups: A Case Study. In Proceedings of International Conference on Enterprise Information Systems, pp. 79-87, 2015.
- [8] G. H. Travassos, F. Shull, M. Fredericks, V. Basili. Detecting defects in object-oriented designs: using reading techniques to increase software quality. ACM SIGPLAN Notices, vol. 34, n. 10, pp. 47-56, 1999.
- [9] R. Cunha, T. Conte, E. S. Almeida, and J. C. Maldonado, A Set of Inspection Techniques Software Product Line Models. In Proceedings of International Conference on Software Engineering and Knowledge Engineering, pp. 657-662, 2012.
- [10] S. Deterding, M. Sicart, L. Nacke, K. O’Hara, and D. Dixon, Gamification: Using Game Design Elements in Non-Gaming Contexts. In Proceedings of the 2011 Annual Conference Extended Abstracts on Human Factors in Computing Systems, pp. 2425-2428, 2011.
- [11] F. Shull, J. Carver, G. H. Travassos, An empirical methodology for introducing software processes. In 9th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 288 – 296, 2001.
- [12] L. Rivero, T. Conte, Improving Usability Inspection Technologies for Web Mockups through Empirical Studies. In Proceedings of International Conference on Software Engineering and Knowledge Engineering, pp. 172-177, 2013.

A Middleware Framework for Leveraging Local and Global Adaptation in IT Ecosystems

Soojin Park

Graduate School of MOT
Sogang University
Seoul, South Korea
psjdream@sogang.ac.kr

Young B. Park

Dept. of Computer Science & Engineering
Dankook University
Seoul, South Korea
ybpark@dankook.ac.kr

Abstract— Impressive advancements in recent smart devices suggests that the direction of software engineering’s future is in development of System of Systems (SoS). Among many concepts that emerged from SoS, we focus on IT Ecosystem – a type of SoS that evolves itself in response to unplanned environment changes. Maintaining autonomy of its participant systems while preserving controllability over entire ecosystem involves various challenges that we need to solve. In this paper, we propose a middleware framework for supporting global adaptation of IT Ecosystem which guides how to determine the optimal adaptation strategy for configuring available systems to satisfy local constraints while achieving global goals. To support the selection of optimal set of participant systems, we have applied genetic algorithm. The effectiveness of our approach is evaluated through analyzing the results of a simulated unmanned forest management IT Ecosystem running the proposed framework while undergoing various environmental changes.

Keywords—self-adaptive systems; dynamic reconfiguration; IT ecosystems.

I. INTRODUCTION

We live among numerous and constant interactions with smart devices running software. Recent technologies such as cloud computing and Internet of Things (IoT) herald a paradigm shift in the operation of software systems, where its focus is transiting from operation of a single system to operation of System of Systems (SoS). A number of recent researches built on the idea of SoS introduced the new concept of IT Ecosystem [2][3]. An IT Ecosystem is a complex system compound composed of interactive and autonomous individual systems, adaptive as a whole based on local adaptivity [1]. Individual component systems within an IT Ecosystem must constantly monitor their environmental contexts in their working territories. If an identified environment change demands reactive change to the system configuration in a participant, that participant dynamically changes its configuration using predefined strategy or knowledge accumulated from previous learnings. The local adaptation loop can be identified as a MAPE-K [3] loop and can be realized through application of adaptation frameworks such as Rainbow [5], MUSIC [6], or DiVA [7].

To create a sustainable IT Ecosystem, we need more than a local adaptation mechanism: we need a means for global

adaptation as to enable IT Ecosystem-wide dynamic reconfiguration in reaction to environmental changes. Unfortunately, existing adaptation frameworks mainly offer benefits limited to local adaptation of a single system, restrictive in their applicability to ensuring sustainability across entire IT Ecosystem. Therefore, in our research, we propose a new adaptation middleware framework designed to support both local and global adaptation mechanisms.

While the proposed framework is to be included in all participant systems, not all components are always run. Among the constituent components, those which implement local adaptation mechanism in response to changes in individual environments are always run. On the other hand, components that execute global adaptation mechanism via deploying new participant systems or dynamically reconfiguring existing systems in response to drastic environmental changes or significant performance drop across the entire IT Ecosystem are only run on participant system with <<Team Leader>> role. The role of *Team Leader* is assigned dynamically, based on the environmental situations of participant systems. The global adaptation mechanism applies a genetic algorithm to make decisions regarding where to place the most appropriate participant system within a working environment, because genetic algorithms can solve computational overhead problems when IT Ecosystems grow in scale.

We have implemented our proposed adaptation framework in a case study of IT Ecosystem for unmanned forest management system, which is among the target domains of our on-going research project. The case study will help understand the benefits, along with the drawbacks, of the proposed adaptation framework. The rest of the paper is as follows: Section 2 introduces the proposed middleware framework for IT Ecosystem adaptation. Section 3 illustrates our proposed mechanisms in use for local and global adaptations within the IT Ecosystem for unmanned forest management. Section 4 discusses the effectiveness of the global adaptation mechanism of the proposed framework through analysis of experiment results. Section 5 reviews related works addressing self-adaptation problems. Finally, in Section 6, we present the conclusions for this study, along with plans for our on-going work.

II. ADAPTABLE MIDDLEWARE FRAMEWORK FOR IT ECOSYSTEM

In this section, we provide an overview of the proposed adaptation middleware framework for IT Ecosystem. The architecture of the framework is composed of five packages: *Felix*, *MAPE Core Bundle*, *Local Adaptation*, *ITE Global Adaptation*, and *ITE Bridge* (as shown in Fig. 1). *Felix* package acts as the bridge between Android platform and OSGi [4] which provides dynamic life cycle management services for components. As depicted in Fig. 1, the proposed framework targets Android platform because of the platform's support for mobility and for its flexibility in its applicability to various application domains where it can control a wide range of passive devices in individual domain's IT Ecosystems.

Felix package includes two major components: *Adaptation Bundle Activator* which invokes bundle service components required to run in higher level packages according to system roles assigned at the participant system's initiation time, and *Configuration Manager* which manages configuration changes when the need for internal component reconfiguration arises due to external environment change. *Effectors* implemented on Android follows instructions given by *Configuration Manager* to carry out the actual configuration change.

MAPE Core Bundle package includes MAPE-K cycle managing components which enables applications to

perceive its environments and determine the next system action to take. Components included in either *ITE Global Adaptation* package and *Local Adaptation* package are architecturally above *MAPE Core Bundle* package and provided as OSGi bundles as to leverage basic component lifecycle management services. All components included in *Felix* package and *MAPE Core Bundle* package are domain independent components. *Local adaptation* package, on the other hand, includes different components depending on what missions individual systems must carry out in order to achieve the global goal of the ITE Ecosystem they participate in. While basic skeleton components for maintaining MAPE cycle operation threads are within *MAPE Core Bundle* package, the domain-specific logic determining what to monitor in order to analyze situational changes and what action to execute are implemented by components included in *Local Adaptation* package. In short, actual adaptation is executed by binding *Local Adaptation* package components to *MAPE Core Bundle* package components.

Information regarding the binding between *MAPE Core Bundle* components and *Local Adaptation* components are stored in *Bundle Registry* within *Felix* package. As participant systems are activated, *Adaptation Bundle Activator* is invoked to look up for information in *Bundle Registry* to check which executable bundle should be bound to *MAPE Core Bundle*, and thereafter activate its corresponding bundles.

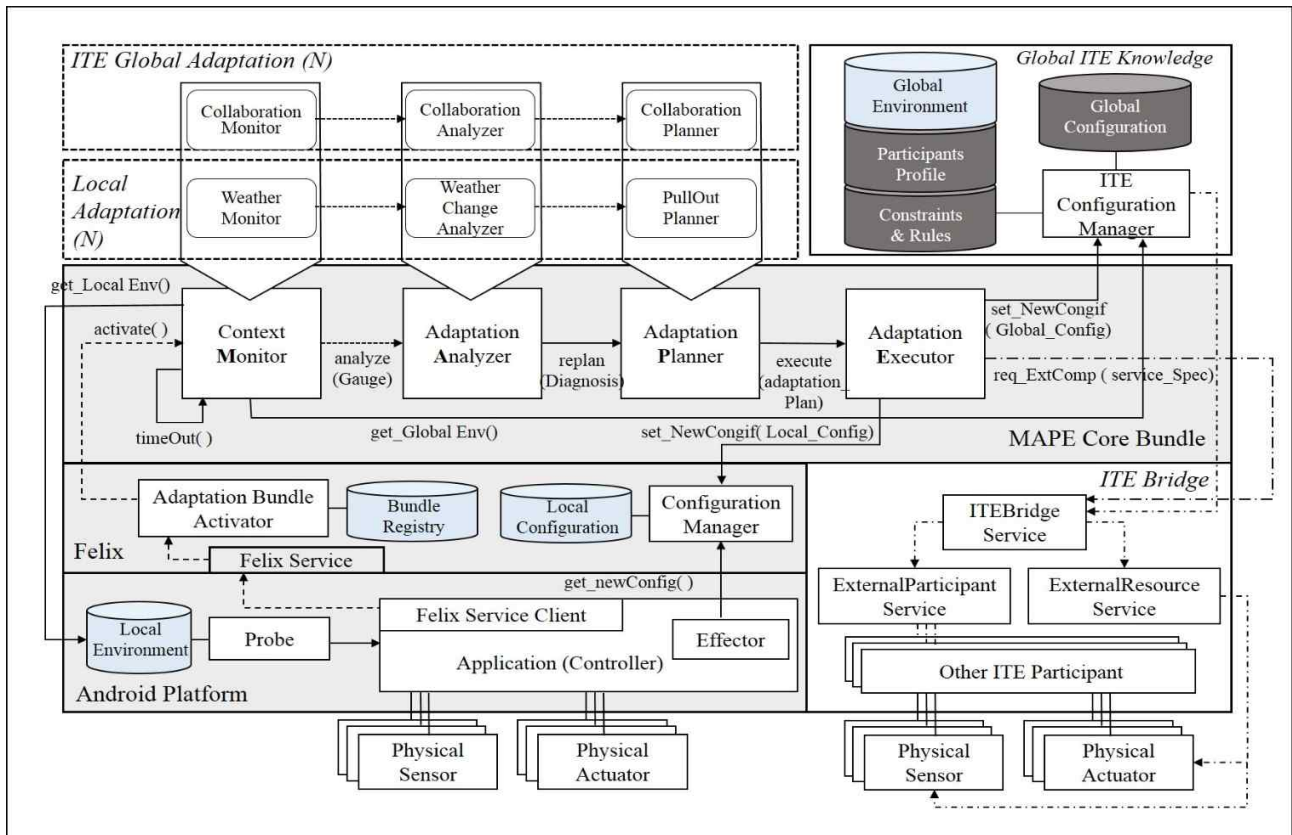


Figure 1. Overview of the proposed adaptable middleware framework for IT Ecosystem

While *ITE Global Adaptation* package is included in all participant systems, it is only run on the participant system assigned with <<*Team Leader*>> role, tasked to monitor the collaboration performance of the entire IT Ecosystem. Where *Local Adaptation* package components provide functional services required to achieve domain goals, *ITE Global Adaptation* package components measure the collaboration efficiency among participant systems executing individual local adaptations, triggering reconfiguration of participant systems when the efficiency is below target threshold.

The core capability of maintaining overall-balanced and sustained service across IT Ecosystem's domain is provided by the *ITE Global Adaptation* mechanism. Details of global adaptation mechanism are introduced alongside a specific case example in the next section. Components in *Local Adaptation* package and *ITE Global Adaptation* package are designed as plug-ins for the four components defined in *MAPE Core Bundle* package. The components within the two packages in Fig.1 illustrate components for unmanned forest management IT Ecosystem, which will be introduced in Section 3. However, the components are replaceable with other components as the target domain changes.

Global ITE Knowledge package stores knowledge to be shared among participant systems and includes *ITE Configuration Manager* which provides the APIs for accessing the knowledge. Knowledge stored at *Global ITE Knowledge* includes environment model which reflects the environment in which IT Ecosystem operates, profiles of participant systems, and constraints or rules that must be considered when mapping participant systems in their working regions. In addition, the model for currently running global configuration is also managed by this package. Such global knowledge can be stored on a separate server or on a cloud storage.

Finally, *ITE Bridge* package handles requests for remote systems when required service components in demand are not within the local system and provides protocols for activating passive devices which are not directly controlled by local systems.

III. A CASE OF ADAPTATION MECHANISM APPLICATION: UNMANNED FOREST MANAGEMENT IT ECOSYSTEM

To demonstrate the proposed framework, in this Section we describe a simulated IT Ecosystem for unmanned forest management (hereafter UFM IT Ecosystem). UFM IT Ecosystem is a management system equipped with twelve unmanned aircrafts, helicopters, and ground vehicles for the purpose of managing nine forest zones each 100 km² in size. Unmanned vehicles assigned to each forest zone utilize sensors and actuators to achieve their goals. In our simulated case, we highlight the process of dynamic reconfiguration within UFM IT Ecosystem in achieving its *Monitor Drought* goal. Depending on the situation, twelve unmanned vehicles are assigned to appropriate roles as to achieve the goal. An unmanned vehicle with <<*Chief Gardner*>> role activates UFM IT Ecosystem's global adaptation cycle to maintain the Ecosystem's sustainability. On the other hand, nine

<<*Surveillant*>> vehicles adjust their driving routes in response to the layout of obstacles in their assigned zones to achieve their goal *Monitor Drought*. In this context, Fig. 2 illustrates an instance of dynamic sequence of adaptation mechanisms in operation where a weather state change triggers a constraint rule violation in a <<*Surveillant*>> role vehicle, causing it to withdraw from its positioned forest zone, leading to local adaptation mechanisms within the vehicle along with global adaptation executed by a <<*Chief Gardner*>> as to select the optimal candidate vehicle for providing sustainable service in the zone.

A. Local Adaptation Mechanism for Dynamic Reconfiguration of Individual Participant Systems

Initially, the forest zone[0][2] has fair weather, has a lake in the zone, and has high forest density. An unmanned helicopter *HE2* has been selected as the appropriate <<*Surveillant*>> for the environment in this zone and is performing *Monitor Draught* goal. We define the paired information of a participant and a zone in the form of (HE2, zone[0][2]) as a chromosome. Such definition becomes useful when genetic algorithm is later applied to select the optimal configuration as part of the global adaptation mechanism.

While *HE2* is carrying out its goal, suddenly a turbulent gale of 25 m/s blows in zone[0][2]. The change in weather is detected by the sensors in the zone and the sensor installed on *HE2*, and is updated to the local environment storage. As *HE2* is assigned a <<*Surveillant*>> role which is not <<*Team Leader*>>, its *MAPE Core Bundle* components are bound with *Local Adaptation* components.

As in Fig. 2 (2), local adaptation is carried out in the following order: *WeatherMonitor* periodically reads sensor data from *LocalEnvironment* storage, calculates gauge values to reflect the current environment zone[0][2], and sends the data to *WeatherAnalyzer* to diagnose if current environment in zone[0][2] violates *HE2* assignment. The diagnose results is passed to *PullOutPlanner* as parameters. *PullOutPlanner* creates a component reconfiguration plan for *HE2* to land safely in a safe region in zone[0][2] and sends the plan to *AdaptationExecutor*. If any component specified in the reconfiguration plan does not exist within the particular system, *AdaptationExecutor* requests it from *ITEBridgeService* by passing the required service features as parameters. *ITEBridgeService* is an OSGi bundle which provides access to external resources. The REST[8] style services provided by *ITEBridgeService* enables the proposed framework to share components, services or other resources among all participants. When all components necessary to land *HE2* have been secured, the results of component reconfiguration is delivered to *ConfigurationManager* within *HE2* as to update its *LocalConfiguration* storage. *Effector* in *HE2*'s controller then reads the updated new configuration and implements the actual component reconfiguration. Lastly, the changed environmental information in zone[0][2] and the service-incapable status of *HE2* are updated to *ITE Global Knowledge* through *ITEConfigurationManager*.

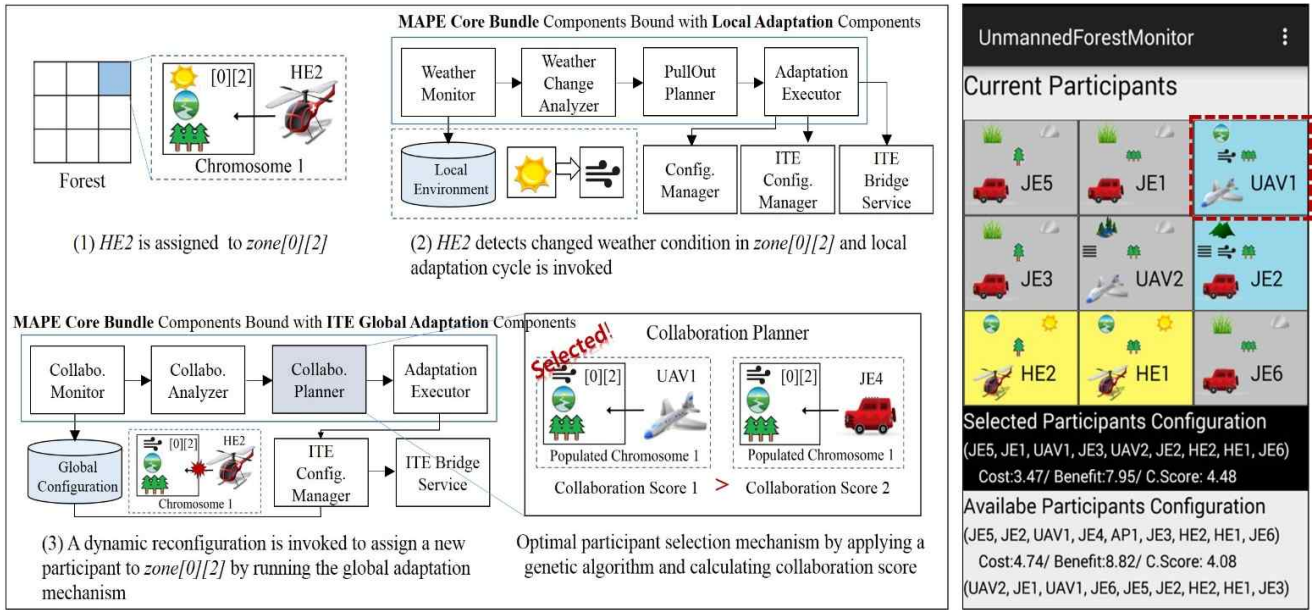


Figure 2. Local and global mechanism by applying the adaptable middleware framework and GUI form of simulated Unmanned Forest Monitor

B. Global Adaptation Mechanism for Dynamic Reconfiguration of the Entire IT Ecosystem

In the given case, AP2 (unmanned airplane) was assigned the role of *Chief Gardner*. As HE2 implements its local adaptation to land during turbulence, AP2 starts to find a new optimal configuration against the changed situation as depicted in Fig. 2 (3). The following process is for the global adaptation for finding a new optimal configuration: *CollaborationMonitor* periodically reads the global configuration and calculates the current configuration's global collaboration score. Global collaboration score is obtained as the sum of all collaboration scores of gene chromosomes (participant and environment zone pair) comprising a configuration. The global collaboration score represents how effective the particular assignment of the selected participant to the zone was. Details of calculating each collaboration score are not presented in this paper. The result of global collaboration score calculation is passed on to *CollaborationAnalyzer* as a gauge value. If the global collaboration score is below a predefined threshold, *CollaborationAnalyzer* identifies gene chromosomes that violate any constraints and passes them to *CollaborationPlanner* as its diagnosis result. Fig. 2 illustrates a case where the gene chromosome (HE2, zone[0][2]) is passed on to *CollaborationPlanner*.

First, *CollaborationPlanner* takes invalid gene chromosomes passed on as a diagnosis result and generates second generation population by mutating the unmanned vehicle information with other possible candidates applicable to the particular zone. Then collaboration scores for the newly generated gene chromosomes are individually calculated, and the chromosome with the highest score is selected and included to the next configuration. Fig. 2 (3) shows a case where the second generation population (UAV1, zone[0][2]) and (JE2, zone[0][2]) have been generated to

replace the invalid gene chromosome (HE2, zone[0][2]). After comparing their collaboration scores, in the next configuration (UAV1, zone[0][2]) will replace (HE2, zone[0][2]) because it has a higher collaboration score. If multiple invalid gene chromosomes have been detected, the above process is repeatedly applied to each invalid gene chromosome as to obtain the optimal configuration with the highest global collaboration score.

Reconfiguring or moving participant systems using obtained optimal configuration requires a plan. In the current example, the change of configuration from chromosome (HE2, zone[0][2]) to chromosome (UAV1, zone[0][2]) implies that HE2 assigned at zone[0][2] must withdraw and UAV1 must move to the zone[0][2]. AP2, now assuming *Team Leader* role, issues orders to other participants using *ITEBridgeService* as to move them sequentially according to the adaptation plan. Then, *ITEBridgeService* activates external *ParticipantService* to implement the operation ordered by *AdaptationExecutor*. Lastly, the new configuration obtained as the result of dynamic reconfiguration is updated to *ITE Global Knowledge*.

The right-most part of Fig. 2 shows the captured GUI form representing the status of nine forest zones in the simulated IT Ecosystem for unmanned forest management, along with the configuration of unmanned vehicles stationed. As the result of the aforementioned mechanisms in operation, we can visually confirm that UAV1 is newly stationed in the forest zone[0][2] which is highlighted by a dashed box.

IV. EVALUATION

Here we look at the performance evaluation results. The main objective of the proposed framework is to provide efficient global adaptation mechanism to guarantee sustainability in entire IT Ecosystem without requiring human intervention. To evaluate the performance, we have

created a UFM IT Ecosystem that simulates continuous weather change (wind velocity, weather type, etc.) to trigger series of global system reconfigurations. Weather changes are divided into two types: slow and rapid. Another element of change introduced is fuel consumption: an internal status of participant systems simulated in correspondence to the distance covered by the participant. Fuel consumption represents fuel efficiency determined for each vehicle.

Graphs in Fig. 3 trace the trends in cost, benefit, and global collaboration scores (*c.score* in the graphs) of optimal configurations selected at every monitoring interval. Cost and benefit factors of each IT ecosystem naturally depend on its corresponding domain. In this case, the required amount of money for operating each unmanned vehicle is calculated as a cost value, and the coverage of drought monitoring work by an individual unmanned vehicle per unit time is calculated as a benefit value. *c.score* is derived from the cost and benefit values and represents the degree of configuration efficiency of the 9 participants in the forest zone; a higher *c.score* indicates a more efficient configuration. Graphs (a) and (c) at the left of Fig. 3 depict the changes in cost, benefit, and *c.score* values in such cases where the proposed dynamic reconfiguration framework is not provided to participant systems that become incapable of continuing *Monitor Draught* goal due to weather changes or fuel shortages. In case of graph (a) where weather changes were mild, *c.score* decrease is found from the 6th monitoring interval. This decrease indicates an event where one or more participants in the forest zone, among 9 total, have become unavailable. As time passes, the number of disabled participants increases dramatically around the 8th monitoring interval, and by the 9th interval all participants have become disabled. Since all participants are disabled at the 14th interval, further monitoring renders no additional information. Therefore, in graph (a), and in all other graphs in Fig. 3, the

scope of trend tracing is limited from the first to the 15th monitoring interval.

Like graph (a), graph (c) depicts the trends in cost, benefit, and *c.score* values when initially positioned participant systems operated statically. However, in contrast with graph (a), graph (c) represents an environment in which weather conditions change more rapidly. As the result, where graph (a) shows gradual trends change, graph (c) shows acute decrease in *c.score* starting from the second monitoring interval where the participants begin to fall into service unavailable status. The point of time when all participant become unavailable remains the same at the 14th interval, but the average *c.score* during 15 monitoring intervals was significantly lower in the weather turbulence in the environment of graph (c), measured at -3.3 which is much lower than -1.09 of graph (a). Rapid weather changes accelerated the occurrence of constraint violations in participant unmanned vehicles, drastically reducing collaboration efficiency among participants.

Unlike graph (a) and (c), the two graphs (b) and (d) on the right side of Fig. 3 show collaboration scores of UFM IT Ecosystem where the proposed global adaptation mechanism is applied. In contrast with (a) and (c) where dynamic reconfiguration is not in effect, it can be seen that measured cost, benefit, and *c.score* values are stable at all monitoring intervals regardless of weather conditions. Initial configuration of participants in each forest zone was identical as graph (a) and (c). Likewise, the first participant to become unavailable occurs on the second interval, as the result of local internal adaptations in each participant that leads to a constraint violation. However, the values captured in graph (b) and (d) indicate that the global adaptation executed by a *Team Leader* participant ultimately ensured sustained service where unavailable participants were replaced with the most appropriate replacement participant.

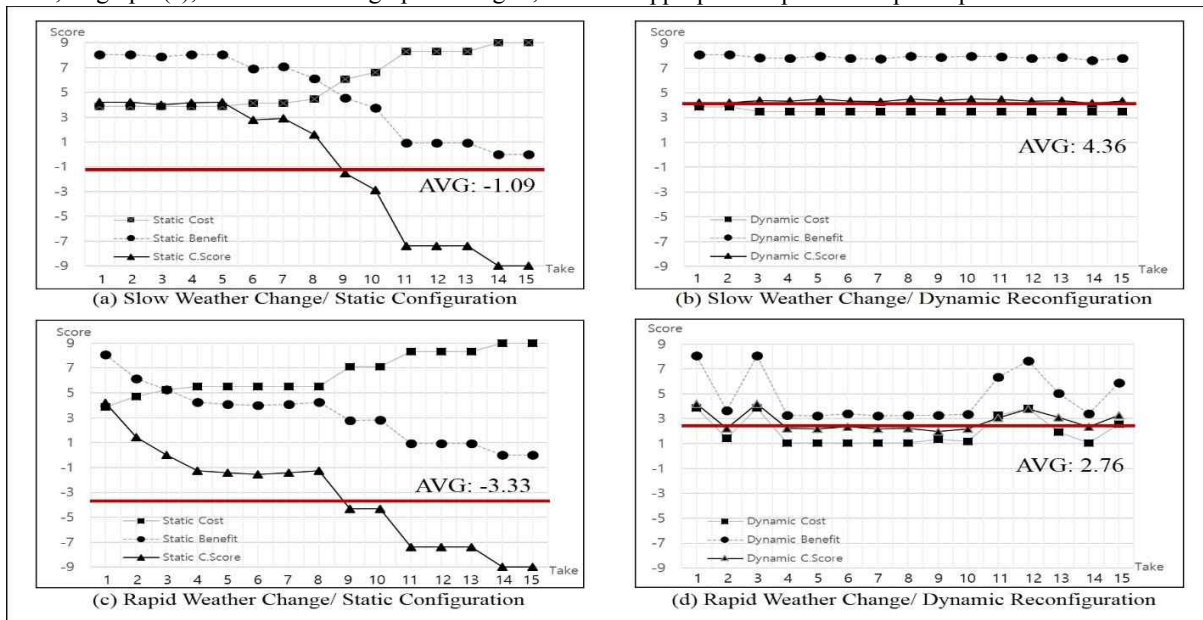


Figure 3. Calculated results of cost-benefit value and collaboration score on selected optimal configuration extracted from each take: (a)(c) without global adaptation mechanism vs. (b)(d) with global adaptation mechanism

In case of graph (b) where weather changes were relatively mild, the value trends are stable without any major fluctuation with average *c.score* at 4.36. This value is significantly higher than the average of -1.09 in graph (a) where no global adaptation cycles were applied. In case of graph (d) where rapid weather changes took place, value changes in cost, benefit, and *c.score* can be observed. However these changes are minor in comparison with graph (c) where global adaptation was not in use: the average in graph (d) is 2.76, considerably higher than -3.3 in graph (c).

Even acknowledging the limited nature of simulated environment test results, it can be safely assumed that the proposed framework's local adaptation and global adaptation mechanism played a positive role in ensuring sustainability of services that are vital in completing the goal of the entire IT Ecosystem.

V. RELATED WORK

There are several frameworks for single self-adaptive systems, such as Rainbow [5], MUSIC [6], and DiVA [7]. Such frameworks are invented to support MAPE-K adaptation control loops. Rainbow [5] framework introduced a reusable infrastructure as to separate concerns between adaptation and application logic, thereby providing architecture-based self-adaptability. While the reusable infrastructure enables self-adaptation with relatively small cost and effort, the Rainbow framework is limited in that its scope supports self-adaptation only in certain situations when situation-specific action rules are applicable. MUSIC [6] combines previous component-based development methods with Service Oriented Architecture (SOA) in that it breaks down all necessary components of self-adaptation into business logic, context awareness, and adaptation concerns as to respond to the distributed and dynamic requirements in mobile environments. However, MUSIC is limited in its need for manual adaptation plan update or replacement because the framework does not include goal management features in its MAPE-K self-adaptation layers. DiVA [7] mainly provides methodologies and framework for developing self-adaptive systems and for managing variability of self-adaptive systems. Its architecture is based on the characteristics of aspect-oriented programming and supports self-adaptation through dynamic addition of appropriate aspects in the form of plug-ins.

While existing works have differentiated benefits, they share the common limitation that self-adaptation is limited to single systems with focus on local adaptation. As their architecture is proposed as conceptual models, developers implementing self-adaptive applications in real-life must rely on their own experiences to find working solutions in their actual environments.

VI. CONCLUSIONS AND ON GOING WORK

In this paper, we have proposed an adaptation framework supporting local adaptation for individual participant system as well as global adaptation across the entire IT Ecosystem. Where existing framework architectures mainly focus on the concept of adaptation, our research details components at a

more concrete level. An IT Ecosystem literally creates an ecosystem composed of individual systems assigned to achieve a common goal even without human intervention. In this context, reconfiguration is a core capability in maintaining overall balance and sustainability in service operation across domains covered by IT Ecosystem. Our work provides optimal configuration in response to environmental changes. Quantitative evaluation shows that the proposed dynamic reconfiguration framework helps IT Ecosystem provide sustainable services even in frequent environmental changes and through successive failures in its participant systems.

In our future research, we plan to continue our designed experiments to quantitatively verify how genetic algorithm reduces the overhead from adaptation cycles to determine optimal configurations. Further, we will continue to self-evaluate as we extend our framework to other domains of IT Ecosystems.

ACKNOWLEDGMENT

This work was supported by the Industrial Convergence Foundation Technology Development Program of MSIP/KEIT [10044457, Development of Autonomous Intelligent Collaboration Framework for Knowledge Bases and Smart Devices] and Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning [No. 2012M3C4A7033 348].

REFERENCES

- [1] A. Rausch, J. Muller, D. Niebuhr, S. Herold, and U. Goltz, "IT Ecosystems: A new paradigm for engineering complex adaptive software systems," In Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on, pp. 1-6, 18-20 June 2012.
- [2] K. Manikas and K. M. Hansen, "Software ecosystems - a systematic literature review," Journal of Systems and Software, vol. 86(5), pp. 1294-1306, 2013.
- [3] IBM Autonomic Computing Architecture Team, "An Architectural Blueprint for Autonomic Computing, Tech.Rep," IBM Hawthorne, NY, USA, June 2006.
- [4] <http://www.osgi.org/Specifications/HomePage>
- [5] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste. "Rainbow: architecture-based self-adaptation with reusable infrastructure," , IEEE Computer, vol. 37(10), pp. 46-54, 2004.
- [6] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. A. Papadopoulos. "A development framework and methodology for self-adapting applications in ubiquitous computing environments," Journal of Systems and Software, vol. 85(12), pp. 2840-2859, December 2012.
- [7] A.Z, M. Araujo, F. Kuiper, D. Valente, J. Wenkstern, R.Z. "DIVAs 4.0: A Multi-Agent Based Simulation Framework," Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on, pp.105-114, Oct. 30 2013-Nov. 1 2013.
- [8] http://en.wikipedia.org/wiki/Representational_state_transfer

A Framework Based on Learning Techniques for Decision-making in Self-adaptive Software

Frank José Affonso, Gustavo Leite
Dept. of Statistics, Applied Mathematics and Computation
Univ Estadual Paulista - UNESP
Rio Claro, SP, Brazil
frank@rc.unesp.br, gustavoleite.ti@gmail.com

Rafael A. P. Oliveira, Elisa Yumi Nakagawa
Dept. of Computer Systems
University of São Paulo - USP
São Carlos, SP, Brazil
{rpaes, elisa}@icmc.usp.br

Abstract—The development of Self-adaptive Software (SaS) presents specific innovative features compared to traditional ones since this type of software constantly deals with structural and/or behavioral changes at runtime. Capabilities of human administration are showing a decrease in relative effectiveness, since some tasks have been difficult to manage introducing potential problems, such as change management and simple human error. Self-healing systems, a system class of SaS, have emerged as a feasible solution in contrast to management complexity, since such system often combines machine learning techniques with control loops to reduce the number of situations requiring human intervention. This paper presents a framework based on learning techniques and the control loop (MAPE-K) to support the decision-making activity for SaS. In addition, it is noteworthy that this framework is part of a wider project developed by the authors of this paper in previous work (i.e., reference architecture for SaS [1]). Aiming to present the viability of our framework, we have conducted a case study using a flight plan module for Unmanned Aerial Vehicles. The results have shown an environment accuracy of about 80%, enabling us to project good perspectives of contribution to the SaS area and other domains of software systems, and enabling knowledge sharing and technology transfer from academia to industry.

Keywords—Self-adaptive software; Reference Architecture; Framework; Learning Techniques; Decision-making.

I. INTRODUCTION

Over recent years, one has observed a significant increase in the complexity of software systems and their computational environments. In general, such systems share functional, nonfunctional, physical, and virtual requirements. The human ability to manage systems has shown as inadequate when their complexity increases. Moreover, involuntary injection of faults has often configured as one of the major causes of system failures (especially in the context of Self-adaptive Software – SaS). In SaS, the design decisions are moved towards runtime to control dynamic behavior and individual reasons of such systems about their states and environments.

Reference Architectures (RAs) refer to a special class of software architecture that have become an important element to systematically reuse architectural knowledge [2], [3]. Thus, in previous work [1], [4] we have proposed Reference Architecture for SaS (RA4SaS) – an architecture that provides a guideline set for SaS development and an automated approach for self-adaptation of the software entities¹ at runtime

¹ From this point onwards, SaS may be also referred to as software entities or simply entities.

without human intervention.

Based on the presented context aimed at improving the quality of development processes for SaS, this paper presents a framework based on learning techniques (classifiers and association rules) [5] and the MAPE-K (Monitor, Analyze, Plan, Execute over Knowledge base) [6], [7] control loop for decision-making in SaS. The main purpose of this framework is to classify and analyze sensory data to autonomously detect and mitigate faults at runtime. Thus, we believe that the needs for systems to interface with human administrators may be reduced, alleviating operational-human costs and, ideally, improving upon existing mitigation techniques. Moreover, based on the preliminary results, we believe that our framework may be used in the knowledge management of other types of software systems. For instance, we have applied this framework in the monitoring and eventual corrections of flight plan for Unmanned Aerial Vehicles (UAVs).

In this context, the primary propose of this paper is to supply the industry with supporting strategies to systematize and automate the functionalities of SaS, contributions from Software Engineering (SE) and Knowledge Engineering (KE) are necessary. Other contributions are: (1) the evaluation of a solution for the problem of classifying and recommending solutions at runtime; (2) a flexible strategy for SaS modeling; and (3) regarding the adaptive module, a feasible strategy to rebuild classifiers and rules from specific points where they were interrupted.

Following the introduction, this paper is organized as follows: Section II presents the background and some related work associated to our study; Section III provides a description of RA4SaS and the framework for decision-making for SaS; a case study designed to validate our approach is presented in Section IV; and finally, Section V summarizes our findings, conclusions, and perspectives for further research.

II. BACKGROUND AND RELATED WORK

This section presents the background (i.e., standard concepts and definitions on SaS and RA) and related work on our study. SaS has specific features in comparison to traditional systems since this type of software system constantly deals with adaptations at runtime, fixing new needs of both users and/or execution environment. Moreover, the SaS development has

boosted self- \star properties in general-purpose software systems, such as self-managing, self-configuring, self-organizing, self-protecting, self-healing, and so on. These properties allow systems to automatically react against users' needs or to respond as soon as these systems meet execution environment changes [3], [8], [9], [10].

RA is a special type of architecture that provides major guidelines for the specification of concrete architectures of a class of systems [11]. Some studies [12], [13], [14], [15] have established different investigations to systematize the design of such architectures, guidelines, and processes. Moreover, the effective knowledge reuse of RA depends not only on raising the domain knowledge, but also documenting and communicating this knowledge efficiently through an adequate architectural description. Commonly, architectural views have been used, together with UML (Unified Modeling Language) techniques, to describe RAs. Considering their relevance as the basis of the software development, a diversity of RAs has been proposed and used, including for (self- \star) software.

As related work, Schneider et al. [16] presented a survey on self-healing systems frameworks. According to these authors, these systems can combine machine learning techniques and control loops to reduce human intervention, since such systems are costly to develop and they can autonomously detect and recover themselves from faulty states. The study presented a classification of self-healing frameworks per three categories (techniques): (i) learning methodology (supervised, semi-supervised, and unsupervised); (ii) management style (bottom-up and top-down); and (iii) computing environment (n-tier traditional, cloud, virtualized, and grid/p2p). In Psaiet & Dustdar [7], a survey on self-healing systems was conducted. This survey showed that the number of approaches for the research on self-healing has been very active. Moreover, a selection of current and past self-healing approaches was addressed, as well as explanations for the origins, principles, and theories of self-healing for such approaches. These two studies provided the theoretical basis for the design of our framework.

Qun et al. [17] stated that architecture-based self-healing approaches were used in the architectural model as basis for system adaptation. Such approaches were based on architectural reflection, and their software architectures are observable and controllable. Cheng et al. [18] purposed a software architecture-based adaptation for grid computing. Technically, the study designed a framework based on a software architectural model. This model allows the analysis of the necessity of adaption in an application, enabling repairs to be written in the context of the architectural model and propagated (applied/designated) to the running system. Zadeh & Seyyedi [19] suggested an architecture based on failure-prediction in architectures based on web services. The main goal of this study is to repair the execution process after detection of a failure. In a similar context, Psaiet et. al [20] developed a self-healing approach that enables recovering mechanisms to avoid degraded or stalled systems. Thus, the study designed VieCure – a framework to support self-healing principles in mixed

service-oriented systems. In this context, one can highlight that the literature has revealed important initiatives for the context of this paper.

III. REFERENCE ARCHITECTURE AND FRAMEWORK FOR DECISION-MAKING

This section presents a brief description of our RA to support the development of SaS [1]. Moreover, our approach addresses a framework based on learning techniques and control loop for decision-making in such systems. This framework is part of the aforementioned architecture, whose main purpose is to support the identification of anomalies (symptoms), and propose solutions (treatments) for SaS at runtime.

A. Reference Architecture: RA4SaS

Figure 1 shows the general representation of our RA4SaS [1]. This architecture is composed of four external modules and a core for potential adaptation (dotted line), which represents an “adaptation bus” of the software entities at runtime in an automated approach. In short, our RA works with a controlled adaptation approach, i.e., the software engineer must insert annotations in each software entity so that the automatic mechanisms in the environment execution can identify the adaptability level of each entity. These levels contain parameters that determine where the new changes may be applied. Thus, when an entity is developed, an automatic mechanism performs a scan process, to inspect if such annotations were correctly inserted. After a validation process, such entities can be stored in the entities repositories (execution environment) so that they may be invoked in future adaptations. Next, a brief description of this architecture is addressed.

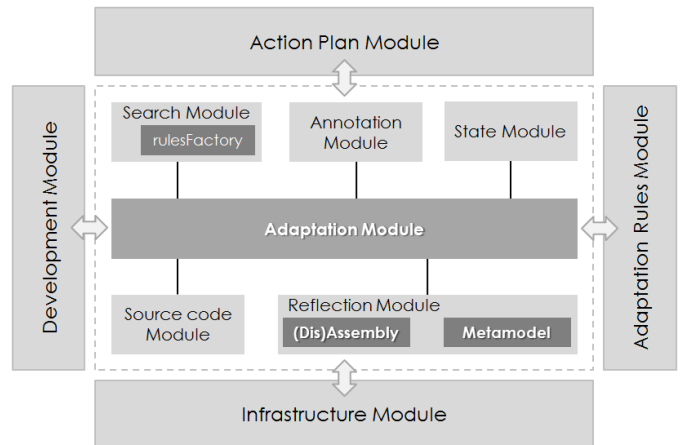


Fig. 1. Reference architecture for self-adaptive software [1]

The **Development Module** provides a guideline set for the development of software entities (SaS). Such guidelines act on requirement analysis, design, implementation, and evolution (i.e., adaptation of the software entities at runtime). The **Action Plan Module** aims at assisting in the adaptation activity of software entities. This module must be able to control:(i) dynamic behavior, (ii) individual reasons, and (iii) execution state in relation to the environment. To do so,

a framework based on learning techniques (classifiers and association rules) [5] and the MAPE-K control loop [6], [21] to support the decision-making of SaS is also part of this module. Section III presents details on the design and implementation of this framework. The **Adaptation Rules Module** provides a rule set (metrics) for adaptation of the software entities. Such rules are stored in the repositories (rule base) and reused when a search for adaptation is performed. The **Infrastructure Module** provides support for software entities adaptation at runtime, i.e., a mechanism set for the dynamic compiling and dynamic loading of software entities. Finally, the core of adaptation represents a logic sequence of well-defined steps so that the adaptation of the software entities is conducted with no human intervention, i.e., all activities of this process are conducted by an automated process as an “assembly line”.

B. Framework for Decision-making

This section presents details of a framework for decision-making in SaS. In short, this framework acts as a non-intrusive supervision modality, i.e., a supervisor system (meta-level) can be coupled to a software entity (base-level) to monitor its internal state of operation or the execution environment in which it is inserted. Besides such supervision modality, this framework incorporates an extension of the MAPE-K control loop and three modules were designed: (i) classification of problems; (ii) recommendation of solutions; and (iii) test of solutions. In addition, sensors and effectors are also components of this framework, since they represent a means of interaction between supervisor system and supervised entities. Sensors are responsible for capturing parameters from the execution environment for the supervised system. Next, the classification module classifies these parameters to identify the changes occurred in each software entity from within the execution environment. Based on this classification and the collected data, an adaptation plan is prepared by the recommendation module to establish a solution for the identified problem. Before it becomes an effective solution, such recommendation must be tested in order to ensure that no “collateral effects” will be propagated to the software system (i.e., other software entities). Effectors deal with the “selected solution” after its testing activities are performed, applying it to the system. Figure 2 shows the control loop utilized in our framework.

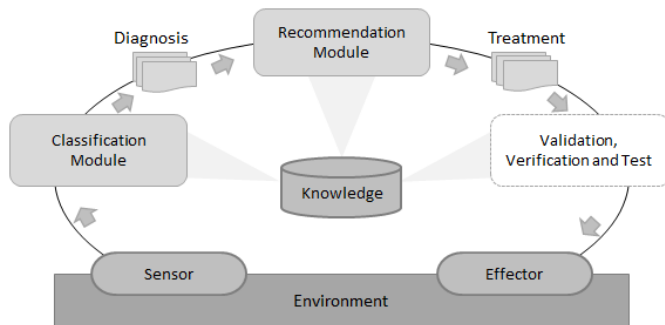


Fig. 2. MAPE-K control loop (Adapted from [6])

Section III-B1 and Section III-B2 present operational details

of the classification and recommendation modules. Due to space limitations, details on the framework testing module are not widely detailed in this paper. However, in short, it is possible to mention that the framework testing module involves a test case selection based on information provided by logs during the system adaption. Section III-B3 provides details on the framework design.

1) *Classification module*: Figure 3 shows the classification module of our framework, whose main purpose is to present a classification for a set of data collected from sensors at runtime. Preliminarily, software engineers must specify the application domain, mapping the “main points” of a software entity (i.e., software system or software architecture) that will be monitored. This specification details the number of attributes of an instance and values that can be assigned to them. Such specification must be stored in a “.arff” file in the “Specification” component and mapped to a database aiming to store all interactions occurred during the execution cycle of our framework. Based on this specification and a set of labeled initial data (Step 1), an incremental classifier is generated (Step 2) in the “Incremental Classifier” component. Next, new data can be collected from the execution environment and sent to this classifier for identification of symptoms (Step 3). Finally, these data are stored in the database as collected and classified (Step 4) after the validation by the recommendation module.

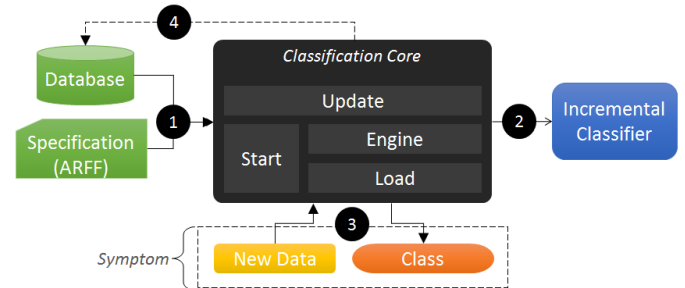


Fig. 3. General representation of the classification module

In the following, we present a brief description of the classification core: (i) **Start**: aims to initialize this module by means of the “Engine” component. As result, an incremental classifier is generated (“Incremental Classifier” component) based on the specification and initial knowledge provided by the specialist; (ii) **Load**: attempts to load data stored in the database, which is organized in two types: (i) specification, i.e., initial knowledge provided by the specialist; and (ii) acquired knowledge, i.e., data obtained during the execution cycle; (iii) **Engine**: represents an abstraction of the incremental classifier algorithm that performs the data classification collected from the execution environment, or from the initial knowledge provided by the specialist; and (iv) **Update**: updates the database after new data has classified. Such update is performed when a message from the recommendation module is received indicating that both data and classification can be stored.

2) *Recommendation module*: The recommendation module supports the selection of an effective treatment for the problem reported in the previous step (classification module). This module has similar operations in comparison to the previous one. In Step 1, an assumption is required: the specification of the domain/problem must be provided by the specialist. To do so, both database of symptoms and domain specification (“Specification” component) must be reused from classification module. Moreover, a treatment database must be created, since it will store the solutions to the problems identified by the previous module. From these databases (symptoms and treatment) and specifications are generated a rule set (Step 2), which intends to map the problems (symptoms) and solutions (treatment). The main purpose for the use of association rule [22] in this module is to detect more significant statistically correlations, via support and confidence, among the symptoms and treatments in order to operate the recommendation of treatments for a symptom set [23]. Why? It is worth noting that there is no interest in a specific attribute (i.e., in a specific treatment), since a symptom set may present some alternatives of treatments. After creation of the rule set, new data (classification module) can be inserted (Step 3) in this module so that one or more treatments may be identified (Step 4). At the end, this module may recommend one or more treatments for the symptom identified. Whether there is more than one treatment, the approach presents a list that must be ordered by the support and confidence criteria of the rules. Thus, one can select these rules one-by-one (i.e., from highest to lowest criterion). After that, the cycle tests the solution as a feasible solving for the identified problem (symptom).

Further, regarding the recommendation module, the internal components of this module (Start, Load, and Update) have the same functionalities as the classification module. Therefore, such components are not presented in the same level of detail. The “Engine” component represents an abstraction for the rule algorithm. Similar to the classification module, other rule algorithms can be coupled to this module as a strategy to compare/evaluate results (i.e., statistical measures).

3) *Framework Design*: Figure 4 shows the main structure of classes of our framework, which is organized in three layers: (i) infrastructure for control loop, containing the core and module packages; (ii) classification and rule algorithms, represented by the algorithms package; and (iii) external resources, represented by the dotted line because they contain a package set developed by third parties. Next, a brief description of these packages in each layer is addressed.

The *core* package contains two interfaces (Observer and Subject) and a class named AbstractObserver. Such interfaces represent the observer and observed roles for the classes of the module package (AbstractModule). Finally, it is noteworthy that the AbstractObserver class implements both interfaces of this package, i.e., implements the methods of the Subject interface and delegates the implementation of the Observer interface for classes inherited.

The *module* package is composed of a class set that represents the MAPE-K implementation (Figure 2) for the

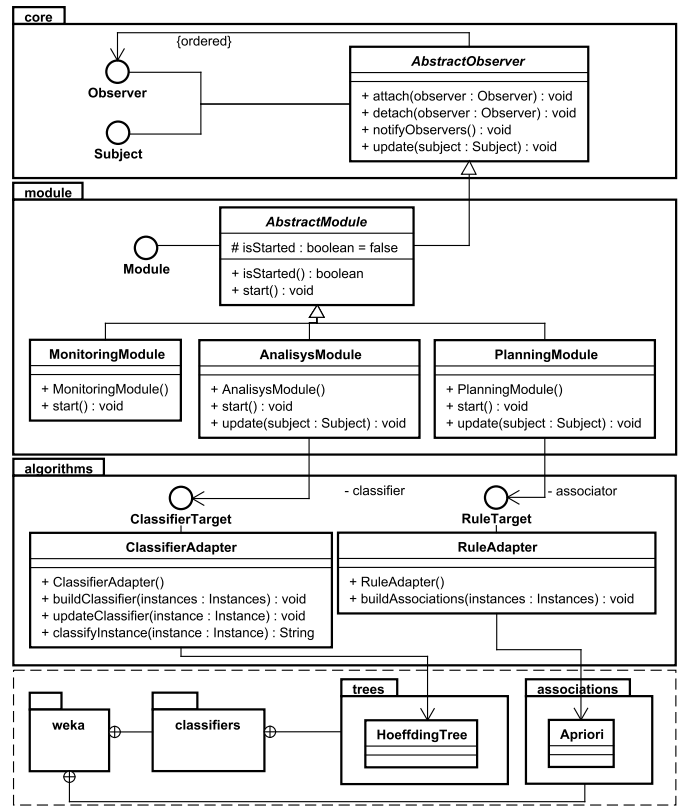


Fig. 4. Framework UML model

classification and recommendation modules. Such implementation is based on the *Observer* design pattern [24], i.e., all classes in this package are, at the same time, a subject and an observer. Thus, three configurations can be created: (i) the `MonitoringModule` class is an observer for the sensors of a software entity and a subject for the `AnalysisModule` class; (ii) the `AnalysisModule` class is an observer for the `MonitoringModule` class and a subject for the `PlanningModule` class; and finally, (iii) the `PlanningModule` class is an observer for the `AnalysisModule` class and a subject for the test module. This strategy enables the classes of this package to be decoupled, acting through event notification by the previous class via a single interface (`Module`).

The *algorithms* package contains a class set that represent the classification and recommendation algorithms. Such classes implement the *Adapter* design pattern [24] so that a common interface is available for both algorithms. The classification module is composed of an interface (`ClassifierTarget`) and a class (`ClassifierAdapter`). This class is an abstraction for the classification algorithm (`HoeffdingTree` class) implemented in the `trees` package. Similarly, the recommendation module was implemented using the aforementioned pattern.

Finally, the packages inside the dotted line represent the concrete classes of our framework, which were developed by third parties. According to *Adapter* design pattern [24],

the ClassifierTarget class is a Target class in the pattern, ClassifierAdapter is an Adapter, and HoeffdingTree is an Adapter. Thus, other algorithms can be coupled to our framework without additional implementation in our system; only the ClassifierAdapter and RuleAdapter classes will be subtly modified. Moreover, the new packages and classes should be represented in the same format as the current ones (dotted lines).

IV. CASE STUDY

To evaluate the applicability, strengths, and weaknesses of our framework this section presents a case study we have conducted. As subject application for our empirical analysis, we have selected an application addressed to the management of an UAV in a simulated environment, as shown in Figure 5. In short, the UAV architecture is organized in three layers: UAV, Communication, and Client. The UAV layer is composed of a UAV set that contains the following components: 3D glasses with radio frequency transmitter; autopilot; navigation camera; day and night vision camera; parachute; solar board; thermal sensor camera and so on. The communication layer contains the servers for communication between UAVs and clients, and time synchronization (NTP – Network Time Protocol). The client layer represents the UAV controllers in different operating systems.

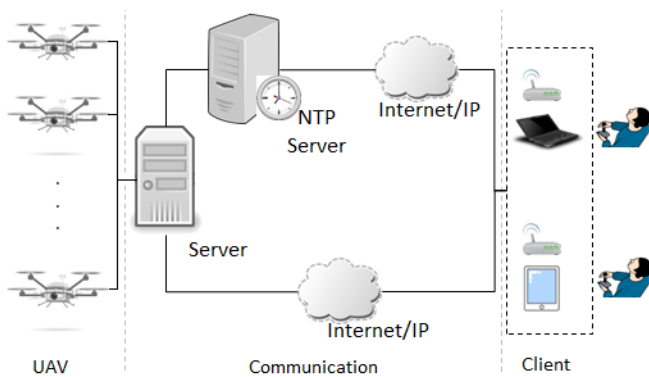


Fig. 5. General architecture for UAV

Operationally, we have instantiated our framework into server (Figure 5), enabling us to collect data from the environment via sensors, and transferring it for classification. In this context, when a problem is detected, a set of useful solutions is presented for correcting the flight plan. In extreme cases, the system may exhibit a recommendation to abort the operation. This last case is recommended when the UAV integrity may be compromised. Then, the UAV location is provided for our system, enabling the vehicle to be rescued. Modifications are made in the flight plan when the collected data tell us that something unplanned is changing in the environment. Thus, even if no decision is taken, the mission of the UAV may be compromised.

The UAVs used in the scope of this empirical study are equipped with seven sensors: (i) altitude and direction, (ii) barometer, (iii) battery level, (iv) humidity, (v) latitude and

longitude, (vi) speed, and (vii) temperature. Some of these sensors provide numerical information that must be discretized, since both algorithms (classification and recommendation) of our framework require data in the form of categorical attributes [5]. Due to space reasons, only one of the sensors was used to show the discretization process. Thus, we have chosen the battery level as our target sensor since it is the power source for all components of an UAV. In addition, the information provided by this sensor represents an estimation of the flight range of the UAVs. Flight range is the time that an UAV can remain flying and, consequently, through this trip autonomy, one can get an estimate on feasible distance of flight. Table I presents the categories for the battery level sensor. The first column shows the range to classify the battery level (second column) on a scale of six to seven percentage points (i.e., A with six points and B and C with seven points). The third column presents a classification in a scale of 20 percentage points. However, it is noteworthy that a classification has three levels, i.e., the A level is the best state of a classification, the B level can be considered as a stability region, and the C level represents a transition stage. Since the discretization process requires a nominal category, we combine the first letter of each classification with respective battery charge levels, as shown in column 4. Finally, it is important to highlight that we have applied the same strategy for the remaining sensors,

TABLE I
CLASSIFICATION FOR THE BATTERY LEVEL SENSOR

Interval	Level	Classification	Class
95 - 100	A		E.A
88 - 94	B	Excellent	E.B
81 - 87	C		E.C
75 - 80	A		G.A
68 - 74	B	Good	G.B
61 - 67	C		G.C
55 - 60	A		R.A
48 - 54	B	Regular	R.B
41 - 47	C		R.C
35 - 40	A		B.A
28 - 34	B	Bad	B.B
21 - 27	C		B.C
14 - 20	A		C.A
7 - 13	B	Critical	C.B
0 - 6	C		C.C

After discretization of the variables, the modeling activity is started. Thus, each sensor will be transformed into an attribute and its respective class in values for this attribute. Next, an initial knowledge must be provided to the databases of symptoms and treatment (i.e., classification and recommendation modules), setting a limitation for our approach. When new data were collected from the environment to be classified by our framework, an environment accuracy rate of 80% was obtained. However, it is noteworthy that although the number may be expressive, this rate can be optimized depending on the initial knowledge provided, since in previous studies this rate ranged from 87 to 94%.

Although no validation process has been used to obtain the results presented in this section, such percentages pro-

vide evidence that the imbalanced data may negatively affect the behavior of both modules (i.e., classification and recommendation). According to our expertise, the following activities must be conducted to overcome such adversity: (i) the domain specialist should conduct the modeling of the problem, i.e., the selection of attributes and values as shown in the discretization process for the battery level attribute; (ii) next, an initial knowledge should be provided so that both modules can be started. It is noteworthy that this knowledge is closely related to the problem and must not be generalized; and (iii) finally, a calibration process of such data must be performed by the specialist, since each problem has specific features and behaviors that should be considered in the execution of both modules. According to [5], [7], [20] this process can optimize the performance of the algorithms of both modules. Finally, we consider the particularity of our subject application as a threat to the validity of our results. Practitioners have been exploring different adaptation rules and creating SaS with different features, limiting the wider generalization of empirical analysis.

V. CONCLUSIONS AND FUTURE WORK

This paper presented a framework for decision-making in SaS. The main contributions of this paper are: (i) SaS area for providing a feasible solution for classification of problems and recommendation of solution at runtime. Our study uses learning techniques as a means to implement the MAPE-K control loop [6], [21]. Moreover, the extension of this loop must be highlighted, since all solutions must be tested before being inserted into the execution environment to avoid that collateral effects regarding to adaptation activity are propagated; (ii) Development facilities with this framework, since an application can be modeled and instantiated without a high level of knowledge by the developers; (iii) Algorithm coupling flexibility, since some applications may require other information or measure for treatment of a problem; and (iv) From operational view point, the reconstruction of classifier and rules for the same point they were interrupted, since all data are labeled as Initial Knowledge (IK) and Acquired Knowledge (AK). Moreover, the databases (symptoms and treatment) are updated when a solution is confirmed as feasible, otherwise the data must be evaluated by the specialist.

As future work, three goals are intended: (i) conduction of more case studies intending to completely evaluate our framework; (ii) evaluation of this framework with other algorithms for both modules classification and recommendation; and (iii) use of this framework in the industry, since it is intended to evaluate its behavior when it is applied in larger real environment of development and execution. Therefore, it is expected that a positive scenario of research, intending to have this framework become an effective contribution to the software development community.

ACKNOWLEDGMENT

This research is supported by PROPe/UNESP and Brazilian funding agencies (FAPESP, CNPq and CAPES).

REFERENCES

- [1] F. J. Affonso and E. Y. Nakagawa, "A reference architecture based on reflection for self-adaptive software," in *SBCARS' 2013*, 2013, pp. 129–138.
- [2] E. Y. Nakagawa, F. Oquendo, and M. Becker, "RAModel: A reference model of reference architectures," in *ECISA/WICSA' 2012*, Helsinki, Finland, 2012, pp. 297–301.
- [3] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *FOSE' 2007*, may 2007, pp. 259–268.
- [4] F. J. Affonso, M. C. V. S. Carneiro, E. L. L. Rodrigues, and E. Y. Nakagawa, "Adaptive software development supported by an automated process: a reference model," *Salesian Journal on Information Systems*, vol. 12, pp. 8–20, 2013.
- [5] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.
- [6] IBM, "An architectural blueprint for autonomic computing," [*Online*], *World Wide Web*, 2005, in <http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf> (Access in 03/13/2015).
- [7] H. Psailer and S. Dustdar, "A survey on self-healing systems: Approaches and systems," *Computing*, vol. 91, no. 1, pp. 43–73, Jan. 2011.
- [8] D. Weyns, S. Malek, and J. Andersson, "Forms: a formal reference model for self-adaptation," in *ICAC' 2010*. New York, NY, USA: ACM, 2010, pp. 205–214.
- [9] L. Liu, S. Thanheiser, and H. Schmeck, "A reference architecture for self-organizing service-oriented computing," in *ARCS' 2008*, U. Brinkschulte, T. Ungerer, C. Hochberger, and R. Spallek, Eds. Springer Berlin / Heidelberg, 2008, vol. 4934, pp. 205–219.
- [10] D. Weyns, S. Malek, and J. Andersson, "On decentralized self-adaptation: lessons from the trenches and challenges for the future," in *SEAMS' 2010*. New York, NY, USA: ACM, 2010, pp. 84–93.
- [11] S. Angelov, P. Grefen, and D. Greefhorst, "A classification of software reference architectures: Analyzing their success and effectiveness," in *WICSA/ECISA' 2009*, 2009, pp. 141–150.
- [12] J. Bayer, T. Forster, D. Ganesan, J.-F. Girard, I. John, J. Knodel, R. Kolb, and D. Muthig, "Definition of reference architectures based on existing systems," Fraunhofer IESE, Tech. Rep., 2004, technical Report 034.04/E.
- [13] E. Y. Nakagawa, R. M. Martins, K. R. Felizardo, and J. C. Maldonado, "Towards a process to design aspect-oriented reference architectures," in *CLEI' 2009*, 2009, pp. 1–10.
- [14] M. Galster and P. Avgeriou, "Empirically-grounded reference architectures: a proposal," in *QoSA-ISARCS' 2011*, New York, NY, USA, 2011, pp. 153–158.
- [15] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Information and Software Technology*, vol. 54, no. 4, pp. 417–431, 2012.
- [16] C. Schneider, A. Barker, and S. Dobson, "A survey of self-healing systems frameworks," *Software: Practice and Experience*, pp. n/a–n/a, 2014.
- [17] Y. Qun, Y. Xian-chun, and X. Man-wu, "A framework for dynamic software architecture-based self-healing," in *ICSMC' 2005*, vol. 3, Oct 2005, pp. 2968–2972 Vol. 3.
- [18] S.-W. Cheng, D. Garlan, B. Schmerl, P. Steenkiste, and N. Hu, "Software architecture-based adaptation for grid computing," in *HPDC-11' 2002*, 2002, pp. 389–398.
- [19] M. H. Zadeh and M. A. Seyyedi, "A self-healing architecture for web services based on failure prediction and a multi agent system," in *ICADIWT' 2011*, Aug 2011, pp. 48–52.
- [20] H. Psailer, F. Skopik, D. Schall, and S. Dustdar, "Behavior monitoring in self-healing service-oriented systems," in *COMPSAC' 2010*, July 2010, pp. 357–366.
- [21] S. Dobson, S. Denazis, A. Fernández, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, Dec. 2006.
- [22] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93, New York, NY, USA, 1993, pp. 207–216.
- [23] C. Tew, C. Giraud-Carrier, K. Tanner, and S. Burton, "Behavior-based clustering and analysis of interestingness measures for association rule mining," *Data Mining and Knowledge Discovery*, vol. 28, no. 4, pp. 1004–1045, 2014.
- [24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

Towards Knowledge-intensive Software Engineering Framework for Self-Adaptive Software

Hyo-Cheol Lee

Dept. of Computer Engineering, Ajou University
NiSE Research Group
Suwon, South Korea
mytion7@ajou.ac.kr

Seok-Won Lee

Dept. of Computer Engineering, Ajou University
NiSE Research Group
Suwon, South Korea
leesw@ajou.ac.kr

Abstract—A self-adaptive system reacts to the changing environment by modifying its functionality in relation to the encountered state of the environment. In order to adapt to a new situation, such system goes through many decision points during the adaptation process. Knowledge forms the basis of decision making within the adaptation process. There are already many existing self-adaptive system frameworks. However, these frameworks have limitation in the way they represent the rationale for adaptation and the semantics behind the knowledge they use. This paper takes a step forward by proposing a knowledge-intensive adaptation framework to both manage knowledge and support the analytical decision making process. The proposed approach represents the adaptation knowledge by using ontology which helps to organize, analyze and extend knowledge. Ontology is able to represent the semantics behind knowledge and provide the evidence for the adaptation. The proposed approach uses a special ontology named the Adaptation Problem Domain Ontology. It specifies the system goals, features, architectures, and the relationship between them. This ontology is used to answer the problem of adaptation at each decision point and determine the appropriate system structure by reasoning the semantics behind knowledge. Thus, the system can consider the semantics behind knowledge for adaptation, and then the stakeholders can understand the adaptation process. We apply the proposed framework to the smart grid domain and show how the system adapts to a new situation using rationale for adaptation and the semantics behind the knowledge.

Index Terms—Self-adaptive system, decision making, ontology, goal model, feature model, role-based architecture

I. INTRODUCTION

As humans interact with changing environments, so a system encounters many different situations which demand different requirements or capabilities. Thus, a system should be able to provide a specific functionality, which is appropriate to the encountered situation, for the user. This has been a great motivator for the development of self-adaptive systems. A self-adaptive system can handle many situations by modifying the system goals, architecture, and functionality in response to changing environments without any human intervention [1]. For self-adaptation, the MAPE-K (Monitor/Analysis/Plan/Execute and Knowledge) process is widely used [2]. Following this process, the system encounters many decision points that DOI reference number: 10.18293/SEKE2015-222

determine what is appropriate in a given situation and achieves the emergent system objectives [3]. At that time, knowledge plays a critical role as the foundation for decision making [4]. Many kinds of knowledge can be used to determine the results and quality of the entire self-adaptation process. That is, the way to use and represent knowledge is important in the self-adaptation.

Many existing self-adaptive system frameworks already regard knowledge as the basis for the adaptation. However, in these frameworks, knowledge is considered as predefined rules, logic and formulas mapping between input and system structures and functionality [5][6][7]. The adaptation process is therefore simplified as mapping between problem and predefined solution. This is similar to the black box testing, where the tester does not consider the internals of the system during testing. This results in the semantics and rationale behind the adaptation to be ignored and implicitly implied. The rationale aspect behind adaptation is essential for stakeholders to understand the reason behind decision making. Existing frameworks are insufficient to illustrate the semantics and rationale behind the adaptation process.

In this paper, we propose the NiSE (kNowledge-intensive Software Engineering) framework for self-adaptive system. The proposed framework adopts an ontological approach to represent knowledge for the adaptation process. Various types of knowledge needed for self-adaptation are systematically organized, connected, and used in the form of ontology. So, using this ontological approach, we are able to provide knowledge-intensive adaptation process including the decision making process which uses the rationale and semantics behind the adaptation [8]. In this adaptation process, the decisions do not just follow predefined logic or formulas as seen in existing approaches [10][11][12], but infer the appropriate ones using the relationship among knowledge. For that, the APDO (Adaptation Problem Domain Ontology) is a key component. APDO is special ontology containing adaptation knowledge such as system goals, features, architecture, and their relationships. During the adaptation, a system infers the appropriate decision using APDO by answering a question at each decision point. It gives support to know which knowledge has been used in the adaptation process. Thus, the proposed approach supports a comprehensive adaptation process through

an ontological approach, and helps stakeholders to understand the rationale and semantics behind the adaptation [9].

This paper is organized as follows: Section 2 introduces the application domain, which is used to illustrate the proposed approach. In Section 3, the proposed approach is described with the help of a case study. We examine other frameworks in Section 4. Section 5 concludes with future works.

II. APPLICATION DOMAIN

In order to verify the applicability of the proposed NiSE framework, we have used a case study in the smart grid domain. A smart grid is next generation electricity grid which simultaneously interacts with demand and supply side using their information [24]. The behavior of a smart grid corresponds with that of a self-adaptive system. The smart grid system also includes and manages many kinds of knowledge such as domain, context, and system structure for adaptation. This provides a domain that is suitable for us to test the feasibility of the NiSE framework.

In this case study, APDO for the smart grid includes the following knowledge: 1) goal model for what a smart grid wants to achieve, 2) feature model to represent variability of a smart grid behavior and component, 3) role-based architecture model which a smart grid can have, and 4) other context and policy related to the smart grid domain. These are correlated with each others and used to make an appropriate decision.

We will use the electricity shortage scenario in this case study [25]. In the smart grid, backup power is stored for emergency situations and should be maintained with certain proportions. Based on the amount of a backup power in a smart grid, there are three states of power warning: *Ready*, *Warning*, and *Severe*. *Ready* is safe state with enough backup power and it maintains its goal and policy. *Warning* is careful state where it needs volunteers to reduce electricity consumption. *Severe* is the most critical state and it is compulsory to regulate electricity consumption. In each state, there is a certain policy to return to the *Ready* state. Thus, maintaining *Ready* state is one of the goals of a smart grid. It means that if the backup power is decreased and the power warning state is changed from *Ready* to *Warning* or *Severe*, a smart grid should change its behavior based on a policy in order to adapt to new situation.

In the case study, we assume that the energy consumptions on the end-users side is suddenly increased due to unexpected weather change. It causes the usage of a backup power to resolve the emergent situation and changes power warning state from *Ready* to *Warning*. A smart grid system monitors these changes and reconfigures its goal, feature, or architecture for return to *Ready* state without any failure. In the next section, the details of NiSE framework is described and explained based on this case study.

III. NiSE FRAMEWORK FOR SELF-ADAPTIVE SYSTEM

NiSE framework for self-adaptive system is proposed to deal with knowledge aspect in the adaptation process and improve the stakeholders' understanding of adaptation by considering the rationale behind the adaptation. The NiSE framework mainly focuses on two perspectives: 1) adaptation

knowledge and 2) associated adaptation process using that knowledge.

A. Adaptation Knowledge Perspectives

In the perspective of adaptation knowledge, we introduce APDO for the knowledge base of the self-adaptation. APDO is a special ontology, which defines knowledge including system structure, rules, and relationships between them [22]. Figure 1 describes the relationships among the various kinds of knowledge. It shows not only the main system structures such as goal, feature, and role architecture, but policy, software engineering process, context, and domain knowledge as well. These kinds of knowledge are used to determine the system behavior and the system architecture. By using these multi-dimensional relationships among many different types of knowledge for the adaptation, we are able to understand the internal process of decision making for the adaptation and support for the stakeholders to comprehensively understand the rationale behind the adaptation.

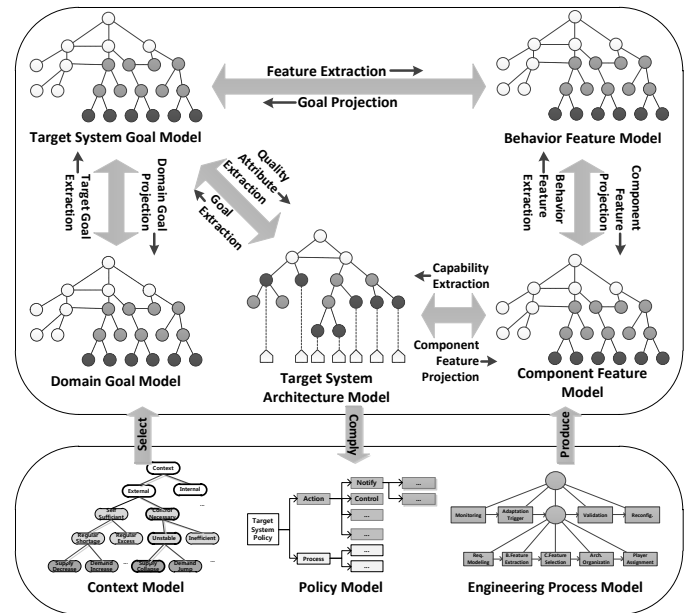


Figure 1 Overview of Adaptation Problem Domain Ontology

Among many types of knowledge, goal, feature and architecture models are directly related to the system structure. As we move from goal models to architecture model, the degree of abstraction is decreased and the details of the system structure are extracted. In order for each model to be associated, we explain the meaning and characteristics of each model.

Goal is the objective that the system wants to achieve [5]. It is used to represent the system's functional and quality requirements [14][15]. In NiSE framework, there are two types of goal model: domain goal model and target system goal model. Domain goal model has all possible goals that a system can achieve. As a subset of domain goal model, target system goal model only includes the goals that the system needs to achieve in given situation.

Goal model is the highest level of abstraction in NiSE framework. When a system goal changes, the purpose of a

system behavior also changes. If a system needs to change its goal, it should find a new goal, which can resolve the problem in a new situation, among domain goal model. Thus, setting domain goal model is defining the available adaptation strategies that a system can have.

Feature is defined as “A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems” [16] and used to represent the system variability and commonality [17]. In NiSE framework, feature model is used not only to represent the variable points at which the system can have diverse options of its functionalities or architectures, but also to reduce the abstraction gap between goal model and architecture model.

For this purpose, NiSE framework includes two types of feature model: behavior and component feature model. Behavior feature model represents atomic actions and component feature model represents functional modules that realize those atomic actions. Behavior feature model is close to goal level and component feature model is close to architecture level. Using these models, a system can connect goal problem space and architecture solution space smoothly and represent variability and commonality with specific articulation [7].

The NiSE framework includes the system architecture using the role-based design approach [18]. It has many advantages to specify adaptive architectural design. Role is the abstract architecture unit, which does not exist in real world. The system is composed with the organization, which is comprised of several roles. The real system components play certain role to make a complete organization. This mapping is separately processed with constructing organization. Therefore, late binding between role and player is possible. It makes loose coupling between the system architecture and real implementation, and flexible architecture to easily change the system components [19][20].

Role model can represent quality requirements of the system through a contract. A contract is the specification of the interaction between roles [21]. A contract includes the process and the measurement. A process describes how the roles interact with each other and the measurement specifies achieving a contract. By measuring the degree of satisfaction of a contract, we can quantify the quality requirements as well.

Furthermore, knowledge of context, policy, software engineering and etc. are able to be represented following diverse models and standards. The form of these kinds of knowledge is not strictly restricted. Furthermore, in APDO, the system engineer can define and add new knowledge.

When a system encounters decision point, including a set of decision questions, the system queries APDO using the above mentioned knowledge as a form of ontology for determining appropriate decision. For example, in order to answer the question “Whether the current structure is in need of adaptation?”, first of all, the system checks whether current system structure is appropriate for a given context or not. For that, the relations between context model and the system structure models are used to infer the answer. If the situation is changed, the system determines the level of adaptation based on the different system structure models and starts adaptation

to satisfy new objective of the current situation by changing its goal, feature, role-based architecture or all of them.

The advantages of APDO are 1) supporting intuitive way to manage adaptation knowledge and 2) providing the evidence of the decision making during the adaptation process. When knowledge is extended and modified, the engineer has a trouble to predict the available situation and architecture based on new knowledge. Thus, it takes a lot of time and effort to infer the available situations and appropriate architecture corresponding to each situation [23]. However, if the engineer uses ontology, the engineer just defines knowledge and relationship among them in ontology. And then, the unpredictable and emergent solutions, which were difficult to determine by human, can be automatically inferred by the system. Besides, because many kinds of knowledge are used, it is able to provide the evidence of the adaptation to understand the rationale and semantics of the adaptation. This enhances the traceability between the situation and the adaptation outcome. It makes the stakeholders understand the adaptation process and application result.

The system engineer or domain experts define APDO, because it needs many kinds of knowledge of the system and domain. The system structures such as goals, features and architecture models have formalized engineering method which helps to define them. All models are converted to ontology classes using those meta-models and the relations between them are represented as object and data properties in ontology. Other knowledge such as policy and context are also illustrated in ontology based on the engineer-defined models. Therefore, the preprocessing of knowledge is required.

B. Adaptation Process Perspectives

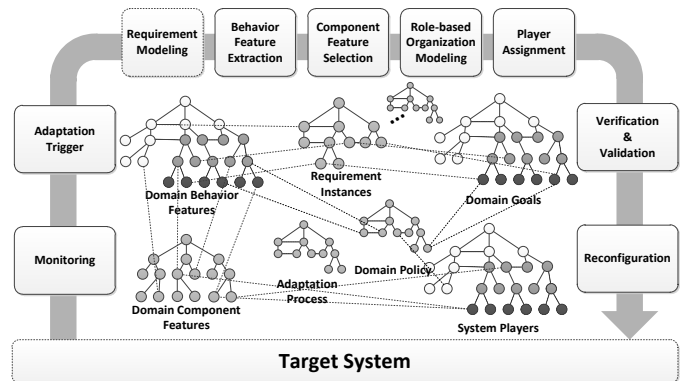


Figure 2 NiSE Adaptation Process

With the purpose of making a proper decision based on knowledge, Figure 2 shows the NiSE adaptation process. The adaptation starts from monitoring the environmental factors to reconfiguring current system architecture into the inferred system architecture. By following this process and answering the adaptation questions using knowledge, a system can make an appropriate decision, and then consequently adapt to the new situation. Each adaptation phase has unique decision points and several adaptation questions for making a decision. For instance, in the scenario described in Section 2, when the weather suddenly changes, the system can raise a question such as “Whether it is needed to adapt?” And then, through

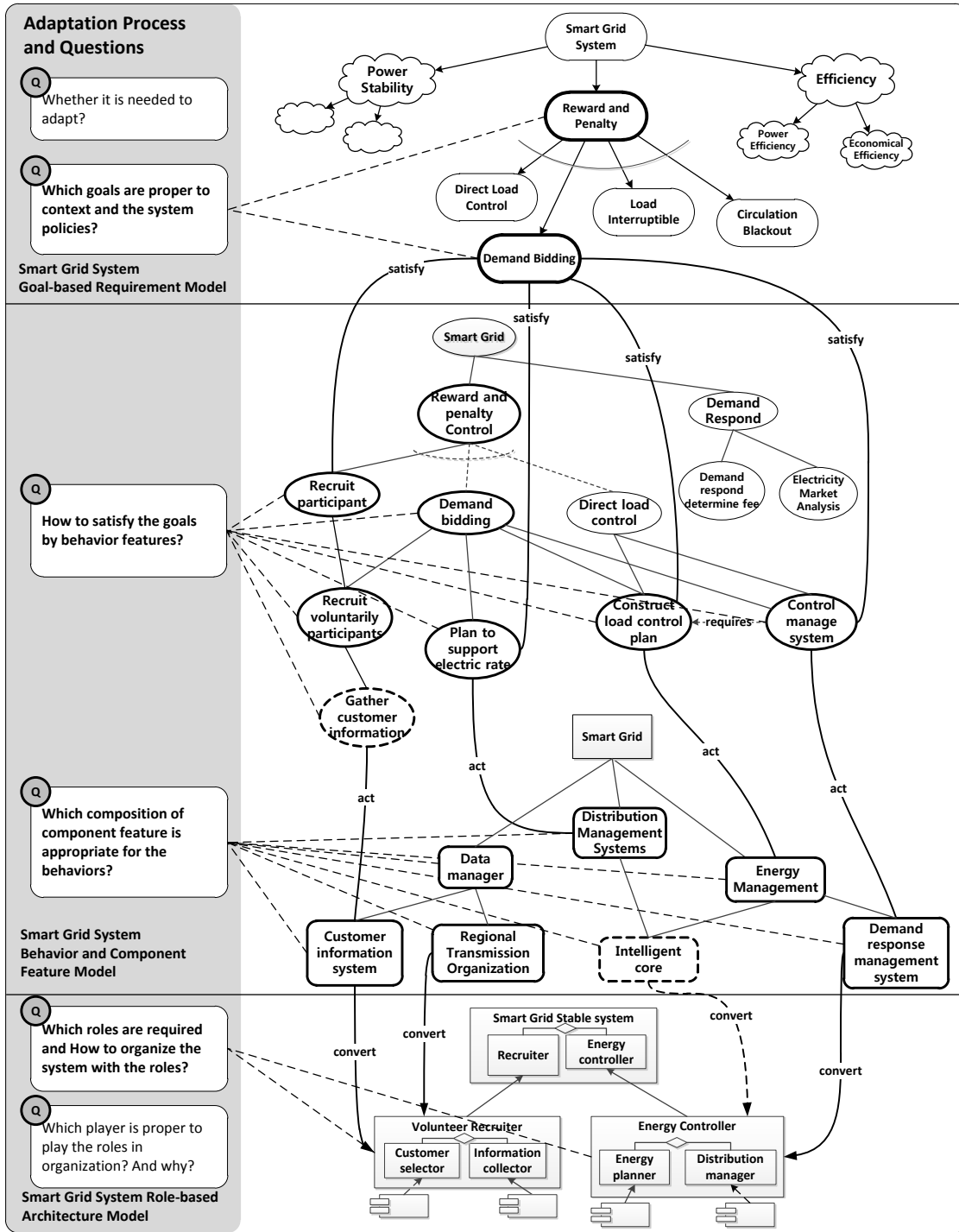


Figure 3 Example of NiSE Adaptation Process Using APDO

knowledge of context, policy and goals and the relationship between them in APDO, the system answers these questions and makes an appropriate decision. These decisions finally affect the system structure in order to satisfy new policies, contexts or requirements.

In order to understand the knowledge-intensive adaptation process, we show a simple example of the adaptation process

using APDO. In this example, we define APDO with 70 classes, 68 object properties, and 25 data properties with respect to the smart grid system's context, policy, goal, feature and architecture to answer the questions shown in Figure 3 in accordance with the scenario in Section 2.

Figure 3 shows the adaptation process with questions at each decision point. Main adaptation process including from

goal-oriented requirement modeling to role-based architecture design is shown based on predefined scenario. The blocks in Figure 3 represent goals, behavior features, component features, and organizations with roles. Each goal is satisfied by behavior features. These behavior features are also performed by component features. Based on selected goals, behaviors and components, which are able to perform the given behaviors, are determined. Lastly, these features are connected to organization and role which are composed of the corresponding capabilities.

In the scenario, energy warning state is changed from *Ready* to *Warning*. To address this change and return to a stable state, the system should increase backup power and decrease current usage of electricity. This is a smart grid domain policy used when energy warning state is changed to *Warning*. For this scenario, APDO includes several knowledge areas such as knowledge of policy, context, and system structure with three abstraction levels (Goal, Feature, and Role-based Architecture) and the relationship among them.

At first, based on the policy, the smart grid system determines that it needs to adapt, and through the defined relationship between *Warning* state and *Demand Bidding* goal in APDO, *Demand Bidding* goal is selected as the proper goal, which are the answers of the first and second questions in Figure 3. *Demand Bidding* goal has *satisfy* relations with *Recruit Participant*, *Plan to Support Electric Rate*, *Construct Load Control Plan* and *Control Manage Systems* behavior features. These relations support that these four behavior features become the answer of the third question. Using act relation between behavior feature and component feature, *Customer Information System*, *Distribution Management Systems*, *Energy Management*, and *Demand Response Management System* are selected as the appropriate component features. It is the answer of the fourth question in Figure 3. In the fifth and sixth questions, these component features are converted to the roles and organization shown in the bottom of Figure 3 via *convert* relation between them, and these roles or organizations are played by the smart grid system components capable to perform them to change its architecture and satisfy the new goal. Consequently, during the adaptation process, these decisions are addressed by answering the questions shown in Figure 3 through APDO and the system changes its goals, features, and architecture [13].

Using the proposed adaptation process, the system makes an appropriate decision with convincing evidences to assure the high quality of the adaptation based on the answer to decision questions. It also support the stakeholders in understanding the rationale behind the adaptation, as they are able to know why the adaptation happens and how it is processed.

IV. RELATED WORKS

In related work, we examine existing self-adaptive system frameworks. We will compare other frameworks with the one proposed in this paper, especially in terms of decision making in the adaptation process and knowledge representation.

Rainbow is a framework for developing customized self-adaptive system [10]. It is composed of two components: managed system and manage system. Managed system is the

system, which directly adapts to the environment. Manage system controls managed system through MAPE-K process. In rainbow, the strategies are defined as the adaptation unit, which a system can take. A strategy contains the system architecture and several tactics. And, it is defined through *Stitch* and *Acme*. Using utility theory, a system is able to quantify which strategy can achieve system objective with the highest utility value. Based on these results, a system selects and changes its architecture that is suitable for new situation.

As mentioned before, the adaptation unit for rainbow is strategy. All the strategies that the system can take are already defined at design time. Thus, a system cannot consider various adaptation problems and provide enough flexibility of the system architecture. However, proposed approach models only knowledge for adaptation and a system infers the appropriate decision using that knowledge at run-time. In other words, the proposed approach does not determine the available architecture, but design knowledge in order to determine appropriate architecture at run-time. It supports high flexibility and traceability by providing the evidence of the adaptation. Proposed approach provides high understanding of the adaptation process to the stakeholders as well.

MADAM (Mobility and Adaptation enabling Middleware) is specially focused on the middleware for the self-adaptation at mobile platform [11]. Through MDA (Model-Driven Architecture), the user defines the system architecture model and the system adapts to new situation by changing the architecture model. In order to select the most suitable architecture model, MADAM uses parameterization, which is a method to apply the external variables to the predefined adaptation formula. The adaptation is processed through functionalized decision making process which means that the situations which the system can face are mapped one-to-one with each architecture model.

MADAM has adaptation middleware to manage system architecture and adaptation process. Thus, the engineer defines this middleware at design time. This adaptation is performed by predefined mapping knowledge, therefore MADAM is not able to consider run-time perspective in the adaptation such as constructing new architecture model, which is more suitable than other defined architecture model. In the proposed framework, we refer MDA approach to represent system architecture with various abstract level, but we infer the adaptation result through knowledge at run-time in order to make an appropriate decision. Namely, we define no direct solution for each situation, but provide knowledge to support decision making process and decide the solution for the system.

DiVA (Dynamic Variability in complex, Adaptive systems) is the framework to support developing the self-adaptive system using AOP (Aspect-oriented Programming) [12]. They use base model and aspect model as the adaptation units for the system adaptation. The base model is designed from the essential components and the aspect model is designed based on the optional component that is able to be added or modified. Simultaneously using both the models, the engineer can easily design the system variability and consider various situations. At design time, not only base and aspect model, but

dependency between aspects in variable points, policy and context are defined as well. Each context and policy is connected to the available aspects, and the system is weaving with base model and selected aspect models at runtime to construct complete system architecture.

In DiVA, the adaptation units are defined at design time as aspect models and it constructs complete system architecture using these models at run-time. It is impossible that a system uses undefined and new aspect for comprising new system architecture. Therefore, it is impossible to consider semantic dependencies when new dependency is defined or many aspects are intertwined. However, proposed approach can manage not only the syntactic relation, but also the semantic relation through ontology by defining knowledge and reasoning the semantics behind that knowledge. It also assures that a self-adaptive system can make a more appropriate decision.

V. CONCLUSION AND FUTURE WORKS

In this paper, we propose a knowledge-intensive software engineering framework for self-adaptive systems. The proposed framework supports the decision making process and the traceability of the adaptation knowledge through knowledge-intensive inference and questions. Thus, the system engineer and stakeholders are able to comprehensively understand the adaptation process and analyze the problem and solution to change non-adaptive systems to be self-adaptive. The limitation of software adaptability is mitigated by extending ontology to add new knowledge for the needed adaptation such as new models or emergent relationships between existing models.

In future works, we need to define an ontology-based software development methodology. In this methodology, the process of knowledge construction about the domain and target system, and the fundamentals for self-adaptation should be defined. Also, inference in decision making process should be extended to resolve uncertainty problems. Uncertainty is an emergent issue in the self-adaptive system. Lastly, the verification and validation of the adaptation framework are needed in order to determine the correctness of knowledge and the decisions made during the adaptation.

ACKNOWLEDGMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2013M3C4A7056233).

REFERENCES

- [1] Betty H. Cheng et al., *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, Software Engineering for Self-Adaptive Systems, Springer, 5525, Lecture Notes in Computer Science, 2009, 1-26.
- [2] IBM, *An architectural blueprint for autonomic computing*, Autonomic Computing White Paper, 2006
- [3] De Lemos, Rogério, et al. "Software engineering for self-adaptive systems: A second research roadmap." *Software Engineering for Self-Adaptive Systems II*. Springer Berlin Heidelberg, 2013. 1-32.
- [4] Kephart, Jeffrey O., and David M. Chess. "The vision of autonomic computing." *Computer* 36.1 (2003): 41-50.
- [5] Yijun Yu, et al. 2008, *From Goals to High-Variability Software Design*, Foundations of Intelligent Systems, Springer, 4994, Lecture Notes in Computer Science (2008), 1-46.
- [6] Brice Morin, et al. 2009. *Taming Dynamically Adaptive Systems using models and aspects*. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, Washington, DC, USA, 122-132.
- [7] Nelly Bencomo, et al. 2008, *Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems*, 2nd International Workshop on Dynamic Software Product Lines.
- [8] Seedorf, Stefan. "Applications of ontologies in software engineering." In *2nd International Workshop on Semantic Web Enabled Software Engineering* held at the 5th International Semantic Web Conference. 2006.
- [9] Gruber, Thomas R. "Toward principles for the design of ontologies used for knowledge sharing?" *International journal of human-computer studies* 43.5 (1995): 907-928.
- [10] David Garlan, et al. "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol.37, no.10, pp. 46- 54, Oct. 2004
- [11] Sebastiano Lombardo, "D.8.9: Mobility and Adaptation enabling Middleware: Final Report", MADAM final report, 2007
- [12] DiVA Project Consortium, "D7.4: A Model-based Approach for Construction and Run-time Management of Adaptive Systems: DiVA practices and Lessons Learned", DiVA White Paper, 2011.
- [13] Brice Morin, et al., 2009, "Models@ Run.time to Support Dynamic Adaptation", *IEEE Computer*, 42, 10, 2009, 44-51.
- [14] Goldsby H.J. et al. 2008. *Goal-Based Modeling of Dynamically Adaptive System Requirements*. International Conference on Engineering of Computer-Based Systems.
- [15] Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, Yijun Yu, 2005, *Towards requirements-driven autonomic systems design*, Proceedings of the 2005 workshop on Design and evolution of autonomic application software
- [16] Kang, Kyo C., et al. *Feature-oriented domain analysis (FODA) feasibility study*. No. CMU/SEI-90-TR-21. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1990.
- [17] Capilla, Rafael, Jan Bosch, and Kyo-Chul Kang. "Systems and Software Variability Management." pp. 25-32, 2013
- [18] Colman, Alan Wesley. "Role oriented Adaptive Design". Swinburne University of Technology, Faculty of Information & Communication Technologies, 2006
- [19] Oreizy, Peyman, et al. "An architecture-based approach to self-adaptive software." *Intelligent Systems and Their Applications*, IEEE 14.3 (1999): 54-62.
- [20] Colman, Alan, and Jun Han. "Roles, players and adaptable organizations." *Applied Ontology* 2.2 (2007): 105-126. Korea Power Exchange, "Power market operating rule", 2013
- [21] Colman, Alan, and Jun Han. "Using role-based coordination to achieve software adaptability." *Science of Computer Programming* 64.2 (2007): 223-245. Korea Power Exchange, "Power market operating rule", 2013
- [22] Lee, S.W. and Gandhi, R. A., *Ontology-based Active Requirements Engineering Framework*, In *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, 2005. IEEE Computer Society
- [23] Daniel M. Berry, Betty H. C. Cheng, Ji Zhang, 2005, *The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems*, In *11th International Workshop on Requirements Engineering Foundation for Software Quality*.
- [24] U.S Department-of-energy. "Grid 2030: a national Vision for electricity' second 100 years". Tech. report, Department of energy, 2003
- [25] Korea Power Exchange, "Power market operating rule", White Paper, 2013

How to Teach the Usage of Project Management Tools in Computer Courses

A Systematic Literature Review

Rafael Queiroz Gonçalves

Department of Informatics and Statistics, Graduate
Program on Computer Science
Federal University of Santa Catarina, UFSC
Florianópolis, Brazil
rafael.queiroz@posgrad.ufsc.br

Christiane Gresse von Wangenheim

Department of Informatics and Statistics, Graduate
Program on Computer Science
Federal University of Santa Catarina, UFSC
Florianópolis, Brazil
c.wangenheim@ufsc.br

Abstract—Project management tools are mandatory to properly manage a software project. The teaching of these tools is carried out in superior computer courses, but often the instructional strategies are used in an ad-hoc manner. This study aims to analyze the literature about teaching of the usage of project management tools and to identify the instructional strategies and the utilized tools. We conducted a systematic literature review to identify the most significant studies that report experiences on this context. After analyzing more than 2700 studies a total of 5 primary studies were selected, and then others were manually included. The instructional strategies and the utilized tools are presented, highlighting the main functionalities and educational features of these tools, as well as the instructional activities carried out to meet the educational goals. Concluding with a discussion of the advances and gaps that remain in this area.

Keywords—Project Management; Project Management Tools; PMBOK; Systematic Literature Review; Teaching; Education.

I. INTRODUCTION

Project Management (PM) is a critical area for many organizations in the software industry. A significant amount of projects still fail due to a lack of proper management, causing problems related to unaccomplished deadlines, budget overrun, or scope coverage [1]. In this context a project is considered a temporary endeavor to achieve a single result, and PM is the use of knowledge, abilities, tools, and techniques that enable a project to reach its goals [2].

Projects problems occur mainly because of the absence of a PM process [3], resulting in a limited control over project restrictions, resources, and stakeholders [1]. The adoption of a PM process may be facilitated by the usage of a PM tool [4]. Despite many organizations still not using any PM tool, the positive contributions that these tools have brought about have increased the interest in their use [5].

The responsibility for the usage of these tools lies with the project manager, who is accountable for the success of the project, having the authority to direct its resources in order to conduct the project in a systematic PM process [2].

Given that the usage of PM tools is not well rooted in organizations, and that projects still fail, a possible cause for this could be the teaching of project managers [1, 6, 7].

The teaching of PM has to address the knowledge on PM, beyond general knowledge on administration, project environment and application area, and interpersonal abilities [2]. However, the teaching of PM should not just be focused on theoretical knowledge, because it is not enough to effectively apply the PM. It is necessary to develop the project manager competencies, which include knowledge (theoretical), abilities (practical), and attitudes (proactivity) [8]. In addition to this, due to the complexity of contemporary software projects, the PM is impracticable without the support of a PM tool, and the usage of these tools is also among the project manager competencies [4]. A PM tool is a software that supports the whole PM process. Among its supported functionalities are: schedule development, resources allocation, monitoring of project performance, etc. [7].

The contribution of this research is the identification of strategies that have been used to teach the usage of PM tools, as well as, the tools adopted. These results may assist teachers in the teaching of this topic, and also assist researchers in the improvement of these strategies by the identification of advances and gaps that remain in this area.

II. BACKGROUND

A. Project Management

The PM conducts the project activities and resources to meet its requirements, since its initiating to its closing (Fig. 1).

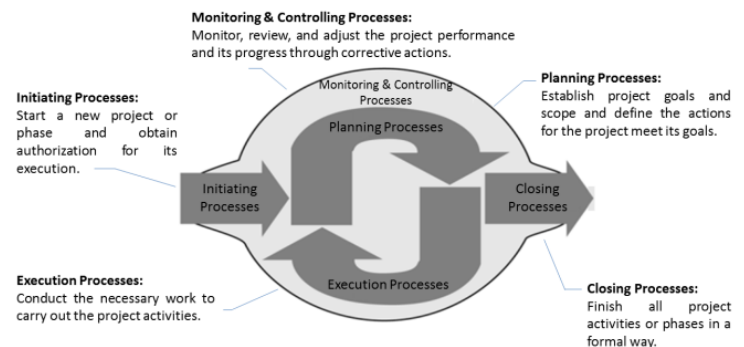


Figure 1. PM processes groups [2].

Orthogonally to these process groups, the PM processes are organized in 10 knowledge areas (Table 1).

Table 1. PM knowledge areas [2].

Knowledge area	Processes to:
Integration	Identify, define, combine, unify, and coordinate PM processes and PM activities.
Scope	Ensure that the project addresses the entire work and meets all its requirements.
Time	Plan, monitor and control the activities that will be carried out during the project so it concludes within the deadline.
Cost	Plan, estimate, and control project costs, so it concludes within the approved budget.
Quality	Define the responsibilities, goals, and quality policies so the project meets the needs that have initiated it.
HR	Organize and manage the project team.
Communication	Ensure the generation, collection, distribution, storage, recovery, and final destination of project information.
Risk	Identify, monitor and control the project risks.
Acquisition	Buy or contract products, services or any resources that are not available as project internal resources.
Stakeholder	Identify and manage the stakeholders and its expectations.

B. PM Tools

A PM tool is a software that supports the whole PM process. Among its supported functionalities are: schedule development, resources allocation, monitoring of project performance, and other functionalities that may support any of PM knowledge areas [4, 7].

Today, there are many PM tools available [9]. These tools are typically classified according to its availability: proprietary (the use of a license or acquisition is mandatory, and it is maintained exclusively by an organization) or open-source (free usage and maintained by the users community). The most relevant proprietary PM tools are: MS-Project (microsoft.com/project) and Primavera (oracle.com/primavera) [4]. Some of most relevant open-source PM tools are: DotProject (dotproject.net), Project.net (project.net), and PhpCollab (phpcollab.com) [10]. The tools also may be distinct by its platform, namely: stand-alone (single user and accessed via desktop) or web-based (multi user and accessed via web browser). Their supported functionalities also vary significantly and may have different approaches, for instance, it may support the whole PM process, just a knowledge area, or, more specifically, just a few activities, as the tracking of worked hours [11].

C. Teaching of PM Tools

The usage of PM tools is part of the project manager competencies [2]. The need of Instructional Units (IUs) for teaching this competency is addressed by the ACM/IEEE reference curriculum for Computer Science [12]. It specifies that students have to develop knowledge in all PM knowledge areas, and have to learn the usage of a PM tool to develop a project schedule, allocate resources, monitor the project activities, etc. Based on these educational needs it is inferred that the usage of a PM tool has to be taught in the application level of the Bloom taxonomy (Table 2), once the knowledge on PM have to be applied through the usage of a PM tool.

Table 2. Bloom taxonomy levels [13].

Level	Refers to the students ability to:
Knowledge	Identify or define some specific information based on previous learning events.
Comprehension	Demonstrate the understanding of an information, and being able to reproduce it by ideas and own words.
Application	Recognize and apply the information to solve concrete problems.
Analysis	Structure the information, fragmenting its parts and establishing their relations and explaining it.
Synthesis	Collect and relate information from various sources, creating a new product.
Evaluation	Make judgments about the value of something (products, ideas, etc.), in relation to known criteria.

Often techniques taught in these IUs include [2, 7, 9]: the Critical Path Method (CPM) – that identifies the project activities that cannot be delayed without affecting the project deadline; the Program Evaluation and Review Technique (PERT) – that calculates the estimated effort to carry out an activity based on three other estimates (worst case, most common case, and best case); the RACI Matrix - describes the participation by various roles in completing project activities; the Resources Levelling - technique in which start and finish dates are adjusted based on resource constraints, with the goal of balancing demand for resources with the available supply; amongst others. To teach these competencies some instructional strategies (Table 3) may be adopted.

Table 3. Instructional strategies.

Instructional Strategy	Description
Direct Instruction	The teacher transmits concepts to students through expositive classes.
Indirect Instruction	The students carry out activities by themselves, and the teacher provides feedback when necessary.
Interactive Instruction	Based on the discussion and sharing of ideas among the students. The teacher acts as a mediator.
Independent Study	Refers to methods which are purposefully provided to foster the development of individual student initiative.
Experimental Learning	Student-centered and oriented to activities. It involves the application of concepts in practical situations.

These IUs also have to evaluate the students learning, and then different kinds of evaluations levels may be adopted (Table 4).

Table 4. Four-level model for evaluation [14].

Level	Evaluation Level	Evaluation description and characteristics
1	Reaction	Evaluates how the participants felt about the training or learning experience.
2	Learning	Evaluates the increase in knowledge or skills.
3	Behavior	Evaluates the degree to which new learning acquired actually transfers to the job performance.
4	Results	Evaluation of the effect on the business environment by the learner.

III. DEFINITION OF SYSTEMATIC LITERATURE REVIEW

The methodology to conduct this research is the Systematic Literature Review (SLR) following the method defined in [15]. A SLR is a study to identify, evaluate and interpret the studies that are available and that are relevant to some research question [15].

A. Research Question

This research aims to identify how to teach the usage of PM tools in superior computer courses. Based on this motivation, we performed a SLR focusing on three research questions:

- a) *RQ1*: Which PM tools are taught in superior computer courses?
- b) *RQ2*: Which instructional strategies are used to teach PM tools in superior computer courses?
- c) *RQ3*: How the instructional strategies effectiveness has been evaluated?

B. Inclusion/Exclusion Criteria

Aiming to select only significant studies, criteria for including/excluding such studies were defined. It had been selected just studies related to the teaching of PM tools, which were published in English language, that are available in digital libraries, and that were published between January 2004 and June 2014. Other criteria restrict the search just for studies that had passed by a peer review process, be it journals or conference proceedings papers. In addition it was excluded: i) Any study that does not use a PM tool (e.g. games, simulators, and e-learning software); ii) Any study that explicitly does not focus on PMBOK (e.g. agile methodologies or other PM approaches), because it is the main reference in area and worldwide accepted [4]; and iii) Any study external to the computer area.

C. Data Sources and Keywords

The data sources had been chosen based on its relevance in software engineering domain, namely: ACM Digital Library, IEEEExplore, ScienceDirect, Scopus, SpringerLink, and Wiley online library. The keywords were defined based on the concepts in the SLR research questions (Table 5).

Table 5. Keywords.

Concept	Keyword and synonymous
Education	Education, teaching, and learning
Project Management	Project management and PMBOK
Tool	Tool, software, and system

IV. SLR EXECUTION

The SLR had been carried out in June 2014. It was conducted by first author, a Computer Science PhD candidate, and it had been reviewed by a senior researcher. The Table 6 presents the amount of returned results by each data source.

Table 6. Returned results by data sources.

Data source	Results
ACM Digital Library (http://dl.acm.org/)	275
IEEEExplore (http://ieeexplore.ieee.org)	1,078
ScienceDirect (www.sciencedirect.com)	65
Scopus (www.scopus.com)	662
SpringerLink (www.springerlink.com)	537
Wiley online library (onlinelibrary.wiley.com)	140
Total	2,757

The returned studies were first analyzed just by their title. The abstract was read only in cases that the titles did not provide evidence of any exclusion criteria. The content of the

study was analyzed only in doubtful cases, for instance, when it was not clear if it was used a PM tool or a simulator. Most studies were excluded because they did not report the usage of any PM tool, but other software (games, e-learning, simulators, etc.). Many other studies were excluded because they are not related to computer area. At the end, just 5 relevant studies were selected (Table 7).

Table 7. Selected studies.

ID	Reference
S1	K. Reid, and G. Wilson, "DrProject: A Software Project Management Portal to Meet Educational Needs," In: Proc. of the Special Interest Group on Computer Science Education, Covington, 2007.
S2	Ž. Car, H. Belani, and K. Pripuzić, "Teaching Project Management in Academic ICT Environments," In: Proc. of the Int. Conf. on "Computer as a Tool", Warsaw, 2007.
S3	G. Gregoriou, K. Kirytopoulos, and C. Kiriklidis, "Project Management Educational Software (ProMES)," Computer Applications in Engineering Education, vol. 21, n. 1, pp. 46–59, 2010.
S4	S. BHATTACHARYA, "Cooperative learning and website in Software Project Management pedagogy," In: Proc. of the Int. Conf. on Interactive Collaborative Learning, Kazan, 2013.
S5	L. Salas-Morera, A. Arauzo-Azofra, and L. García-Hernández, "PpcProject: An educational tool for software project management," Computers & Education, vol. 69, n.1, pp. 181–188, 2013.

Aiming to find more relevant studies, the state of the art section of the selected studies was analyzed, and 3 more relevant studies were found. Although some of these presented tools did include simulation/game features, when analyzing their functionalities it became evident that they may in fact be characterized as PM tools.

Table 8. Manually included studies.

ID	Reference
S6	A. Shtub, "Project management simulation with PTB project team builder," In: Proc. of the 2010 Winter Simulation Conference, Baltimore, 2010.
S7	F. Deblaere, E. Demeulemeester, and W. Herroelen, "RESCON: Educational Project Scheduling Software," Computer Applications in Engineering Education," vol. 19, n. 1, pp. 327-336, 2009.
S8	M. Vanhoucke, V. Vereecke, and P. Gemmel, "The Project Scheduling Game," Project Management Journal, vol. 36, n. 1, pp. 51-59, 2005.



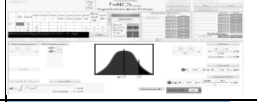
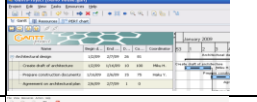

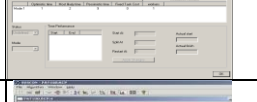

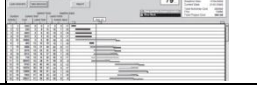
V. DATA SYNTHESIS AND EXTRACTION

After selecting the studies, their data were systematically extracted. The metadata to be extracted from studies were defined based on each research question:

- a) *RQ1*: tool name, classification (availability (proprietary or open-source), platform (desktop or web-based) and propose (general usage or educational)), main functionalities, educational features, print screen.
- b) *RQ2*: addressed process groups and knowledge areas, educational goals, taught functionalities, instructional strategies and activities, students evaluation method, discipline hours.
- c) *RQ3*: evaluation goals, instrument for data collection, sample size, evaluation method and evaluation level.

Firstly, the general features of PM tools are presented in Table 9. As the studies itself do not necessarily indicate these information explicitly, some of the information has been inferred based on the presented reports.

Table 9. General features of PM tools (RQ1).

ID	Tool name	Classifications	Main functionalities	Educational features	Print screen
S1	DrProject	Open-source, web-based and educational.	Tickets creation (analogue to project activities creation and human resources allocation), mailing lists for project communication, and wiki for organizing the project documentation.	-Mailing lists to facilitate the project communication between team members and the teacher. -The forms contain only the strictly necessary fields in the context of the discipline.	
S2	MS-Project	Proprietary, desktop, and general usage.	Schedule development, project team definition, hour/rate configuration for human resources, project progress update and monitoring, baselines control.	Does not apply.	
S3	ProMES	Open-source, desktop, and educational.	Supports the application of CPM, PERT, and RACI matrix techniques.	-Provides scenarios and difficult levels to apply the CPM, PERT, and RACI matrix techniques. -Configuration of experience levels: trainee (student has support of the tool) and professional (no help is provided), and tutorial video.	
S4	Gantt project	Open-source, desktop, and general usage.	Schedule development, project progress updating and monitoring.	Does not apply.	
S5	PpcProject	PpcProject: Open-source, desktop and educational.	Schedule development, support the CPM, PERT, and resources levelling techniques.	The historic of all calculi are maintained on screen for the student follow the calculation procedure.	
S6	Project Team Builder - PTB	Proprietary, desktop, and educational.	Work packages definition, schedule development, and effort, resources, and cost estimations.	Provides scenarios to simulate the execution of a project plan, requiring the students to take decisions which respect the project restrictions.	
S7	RESCON	Open-source, desktop, and educational.	Schedule development, resources allocation, and CPM.	- What-if analysis for the students evaluating the effects of resources inclusion in the project. - Simulation of different schedule development algorithms that solve resource constraint problems.	
S8	Project Scheduling Game – PSG	Proprietary, desktop, and educational.	Schedule development, resources allocation, cost planning, and CPM.	Simulation of project execution requiring the students to take decisions regarding the time/cost trade-off.	

Information related to the instructional strategies for teaching of PM tools usage (RQ2) are presented in Table 10. As the studies itself do not necessarily indicate these

information explicitly, some of the information has been inferred based on the presented reports.

Table 10. Data related to the instruction strategies (RQ2).

ID	Process groups	Knowledge areas	Educational goals	Taught functionalities	Instructional strategies and activities	Students evaluation method	Discipline hours
S1	Initiation, Planning, Execution, Monitoring & Controlling, Closing.	Time, HR, and communication.	After the classes about PM tool usage the students have to use a PM tool to setup a project, create its plan, and keep its progress updated while executing it.	Schedule development, organization of documentation, project communication.	Classification: Experimental Learning Activities: Elaboration of a project plan using a PM tool, and execution of the planned project in groups of students.	Not informed	Not informed. *It had 7 weeks of duration.
S2	Initiation, Planning, Execution, Monitoring & Controlling, Closing.	Time, HR, and communication.	After the classes about PM tool usage the students have to use a PM tool to setup a project, create its plan, and keep its progress updated while executing it.	Schedule development and monitoring.	Classification: Experimental Learning Activities: Elaboration of a project plan using a PM tool, execution of the planned project in groups of students, and production of project artefacts during its life cycle.	-Delivery of exercise carried out using the PM tool. -Theoretical test of objective questions.	Not informed *It had 7 weeks of duration.
S3	Planning	Time, and HR.	After the classes about PM tool usage the students have to use a PM tool to apply the CPM, PERT, and RACI matrix techniques.	PERT, CPM and RACI matrix techniques.	Classification: Experimental Learning Activities: Resolution of problems using CPM, PERT, RACI matrix techniques. For each technique are carried out exercises with ascending difficulty level.	Delivery of problems resolution.	Not informed
S4	Initiation, Planning, Execution, Monitoring & Controlling, Closing.	Scope, time, and HR.	After the classes about PM tool usage the students have to use a PM tool to setup a project, create its plan, and keep its progress updated while executing it.	Schedule development.	Classification: Experimental Learning Activities: Elaboration of a project plan using a PM tool, and execution of the planned project in groups of students.	- 10 minutes project presentation; - Theoretical test of objective questions.	40 hours *20 meetings of 2 hours duration.
S5	Planning	Time, and HR.	After the classes about PM tool usage the students have to use a PM tool to apply the CPM, PERT, and resources levelling techniques.	CPM, PERT, and resources levelling techniques.	Classification: Experimental Learning Activities: Resolution of sequential problems with ascending difficulty levels, involving the application of CPM, PERT, and resources levelling techniques.	Delivery of problems resolution.	4 hours *2 meetings of 2 hours of duration.

S6	Planning, Execution, Monitoring & Controlling.	Scope, time, HR, cost.	After the classes about PM tool usage the students have to use a PM tool to schedule development, HR allocation, and to analyze monitoring and controlling reports.	Schedule development and HR allocation.	Classification: Experimental Learning Activities: Elaboration of a project plan using a PM tool, and management of HRs during the simulation of the project execution.	Not informed	1 hour
S7	Planning	Time, HR.	After the classes about PM tool usage the students have to use a PM tool to develop a project schedule using strategies to solve resource constraint problems.	Schedule development, HR allocation and HR levelling.	Classification: Experimental Learning Activities: Definition of project activities, and its estimations for effort and resources. Execution of different algorithms for schedule development and comparison of their results.	Not informed	Not informed *It was used during a semester.
S8	Planning, Execution, Monitoring & Controlling.	Time, HR, cost.	After the classes about PM tool usage the students have to use a PM tool to schedule development, HR allocation, and to analyze monitoring and controlling reports.	Schedule development, CPM and HR allocation.	Classification: Experimental Learning Activities: Elaboration of a project plan using a PM tool, and management of HRs during the simulation of the project execution.	Punctuation provided by the educational PM tool, based on project completion and its total cost at ending.	2 hours

Lastly, the data related to the evaluation of instructional strategy effectiveness (RQ3) are presented in Table 11.

Table 11. Data related to instructional strategy evaluation (RQ3).

ID	Evaluation goal	Instrument for data collection	Sample size	Evaluation method	Evaluation level
S1	Evaluate if the students are able to manage and carry out projects systematically with the support of a PM tool.	Observation and PM tool database (to identify the PM tool usage pattern by tickets and wiki records).	Not informed. *Superior to 25	Subjective observation in an ad-hoc manner.	Reaction
S2	Evaluate if the students succeed to accomplish projects according to defined processes and using appropriate PM tools.	- Observation. - Students oral presentation.	130	Subjective observation in an ad-hoc manner.	Reaction
S3	Evaluate the students learning of CPM, PERT, RACI matrix techniques through the usage of an educational PM tool.	Observation and students feedback.	20	Subjective observation in an ad-hoc manner.	Reaction
S4	Evaluate if the students are able to prepare and to present a project plan with the support of a PM tool.	Written test and questioner	47	Evaluation of students grade in the discipline, and questionnaire answers.	Learning
S5	Evaluate among PpcProject and MS-Project PM tools, which one is more appropriate for educational proposes.	Questioner	54	Each student has answered twice a questionnaire. The first time about his experience when carried out a few PM activities using PpcProject, and the other after doing the same with MS-Project.	Reaction
S6	Evaluate if the students are able to manage resources in a project respecting its constraints with support of a PM tool.	Observation and students feedback.	Not informed.	Subjective observation in an ad-hoc manner.	Reaction
S7	Evaluate the students understanding about the CPM and schedule development algorithms through the usage of an educational PM tool.	Observation and students feedback.	121	Subjective observation in an ad-hoc manner.	Reaction
S8	Evaluate if the students are able to manage resources in a project respecting its constraints with support of a PM tool.	Observation and students feedback.	Not informed.	Subjective observation in an ad-hoc manner.	Reaction

VI. DISCUSSION

A discussion based on the extracted data of the SLR is carried out aiming to answer the research questions.

In relation to the PM tools that are taught (RQ1), it had been observed that the MS-Project is the most utilized tool. In part it is because the students familiarity with MS-Office environment and also by its availability on university labs. However, many studies (S1, S3 and S5) points out the lack in this tool for some PM processes, as well, the absence of educational features. In an effort to cover this lack there had been developed educational PM tools, such as DrProject, ProMES, and PpcProject. These tools provide educational features, for instance, the configuration of difficulty levels, profiles for student assistance (step by step explanations) and tutorial videos. In addition, the PM tool PpcProject was compared to MS-Project, demonstrating to be as complete as in relation to the supported functionalities, but superior in educational aspects.

When analyzing the instructional strategies for teaching the usage of PM tools (RQ2), it is observed that in all cases it is classified as experimental learning, because involves the usage

of a PM tool during practical classes. Just few studies have reported that some explanation about the PM tool usage is provided before the students start to use it. In other cases, the students need to learn about the PM tool by the exploratory analysis of its functionalities. It also was observed that the time management knowledge area was the most addressed. The HR management was the next most addressed, mainly due to the HR allocation process. It was identified three main kinds of instructional strategies: The first one is related to the execution of practical projects (students organized in groups, build a software and use a PM tool for planning and monitoring it) (S1, S2, S4); The second one focuses on the application of specific techniques, such as CPM and PERT (S3, S5, S7). In this case the instructor presents problems to the students and they work for its resolution using a PM tool. The first strategy covers, at least minimally, all PM process groups, while the second one covers just the planning process group. The last strategy is focused on the management of project resources during the simulation of project execution (S6, S8), requiring the students to make decisions based on the analysis of project monitoring and controlling reports. About

the discipline hours, the first strategy requires more than others, because it includes the project execution, instead of just the application of specific techniques.

Regarding the evaluation of the effectiveness of these instructional strategies (RQ3), all studies reported at least a subjective evaluation, normally in an ad-hoc manner, based on the authors opinion and in a few cases also the students feedback. The evaluations have concluded that the instructional strategies assist in the learning of PM concepts and prepare the students for the professional career. Some more systematic evaluations were carried out in S4, evaluating its effectiveness based on the students grade, and S5 have applied a questionnaire for students to identify their learning experience. Yet, in most cases the evaluations were classified in the reaction level, with focus on the students' perspective.

It was evidenced that the teaching of PM tools assists the students in the comprehension of PM concepts and provides opportunities to the students to have practical experiences through the application of concepts. However, it was noticed that the instructional strategies are too focused on time and RH management, minimally addressing other PM knowledge areas. None of the studies addressed risk management, quality management, acquisition management and others. In part it may be justified by the lack of support of the PM tools to these knowledge areas. Hence, it is evidenced that the developed IUs for teaching the usage of PM tools does not contain instructional strategies that cover the whole PM process, and the gaps still existing in this area are highlighted.

A. Threats to Validity

A common threat in any SLR is the bias inherent to scientific publications that in most cases reports the successes of the experiences, and not its failures. This threat may have hampered the identification of ways to measure the effectiveness of a certain instructional strategy. It was mitigated including a research question to identify how the instructional strategies were evaluated. During the search process the main threat is to not find relevant studies. A migration for this threat includes the use of synonyms for all search keywords. On the other hand, it returned a large amount of results. For instance, the synonyms for the concept of tool bring studies focused on e-learning, games, and simulators. Other mitigating actions included the usage of many data sources, in addition to the manual inclusion of studies based on the state of the art sections of those selected. In the SLR selection phase, the identified threat is related to the influence of the researchers personal opinion. It was mitigated by registering the exclusion criteria that motivated the disposal of each study considered irrelevant, and by the discussion of the results among the SLR participants. This threat also impact on data analysis phase, because some information are not explicit in the studies, and have been inferred by authors.

VII. CONCLUSIONS

This work aims to identify which instructional strategies are been adopted in the teaching of PM tools usage in superior computer courses. To reach this goal, it was carried out a SLR, identifying the most relevant studies in the area. The results

show that, typically, the teaching of PM tools usage is carried out in practical classes and the instructional strategies varies from specific problems resolution or planning a software project. The educational goals in general are focused on the teaching of time and HR management, minimally or not addressing other PM knowledge areas. In part it may be justified by the lack of support of the PM tools to these knowledge areas. Hence, despite the efforts, it is evidenced that the teaching of PM tools usage still does not cover the whole PM process, which is essential for a more efficient PM. Future work may suggest other instructional strategies to fill these gaps, through the adoption of a systematic PM process that covers all knowledge areas, and the usage of a PM tool aligned to such a process.

ACKNOWLEDGMENT

This work was supported by the CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico – www.cnpq.br), an entity of the Brazilian government focused on scientific and technological development.

REFERENCES

- [1] The Standish Group, *Chaos Manifesto 2013*, Boston, 2013.
- [2] PMI – Project Management Institute, *A Guide to the Project Management Body of Knowledge*, 5. ed., Newtown Square, 2013.
- [3] M. Keil, A. Rai, and J. Mann, "Why software projects escalate: The importance of project management constructs," *IEEE Transactions on Engineering Management*, vol. 50, n.3, pp. 251–261, 2003.
- [4] R. Fabac, D. Radošević, and I. Pihir, "Frequency of use and importance of software tools in project management practice in Croatia," In: *Proc. of 32nd Int. Conf. on Information Technology Interfaces, Cavtat*, 2010.
- [5] H. Cicibas, O. Unal, and K. Demir, "A comparison of project management software tools (PMST)," In: *Proc. of the 9th Software Engineering Research and Practice, Las Vegas*, 2010.
- [6] T. Lethbridge, J. Diaz-Herrera, R. Leblanc, and J. Thompson, "Improving software practice through education: Challenges and future trends," In: *Proc. of Future of Software Engineering, Minneapolis*, 2007.
- [7] Ž. Car, H. Belani, and K. Pripuzić, "Teaching Project Management in Academic ICT Environments," In: *Proc. of the Int. Conf. on computer as a tool, Warsaw*, 2007.
- [8] L. Spencer, and S. Spencer, *Competence at Work: Models for Superior Performance*, 1st ed. John Wiley & Sons, 1993.
- [9] L. Salas-Morera, A. Arauzo-Azofra, and L. García-Hernández, "PpcProject: An educational tool for software project management," *Computers & Education*, vol. 69, n. 1, pp. 181-188, 2013.
- [10] A. Pereira, R. Gonçalves, and C. Wangenheim, "Comparison of open source tools for project management," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, n. 2, pp. 189-209, 2013.
- [11] C. Wangenheim, J. Hauck, and A. Wangenheim, "Enhancing open source software in alignment with CMMI-DEV," *IEEE Software*, vol. 26, n. 2, pp. 59-67, 2009.
- [12] ACM, and IEEE Computer Society, *Computer Science Curricula 2013*, 2013.
- [13] B. Bloom, *Taxonomy of educational objectives: The classification of educational goals*, 1st ed. Longmans, 1956.
- [14] D. Kirkpatrick, and J. Kirkpatrick, *Evaluating training programs: The four levels*, 4th ed. Berrett-Koehler Publishers, 2012.
- [15] B. Kitchenham, "Systematic literature reviews in software engineering – A tertiary study," *Information and Software Technology*, vol. 52, n. 1, pp. 792-805, 2010.

Soft Skills in Scrum Teams

A survey of the most valued to have by Product Owners and Scrum Masters

Gerardo Matturro

Software Engineering Department
Universidad ORT Uruguay
Montevideo, Uruguay
matturro@uni.ort.edu.uy

Carina Fontán

Software Engineering Department
Universidad ORT Uruguay
Montevideo, Uruguay
cpfontan@gmail.com

Florencia Raschetti

Software Engineering Department
Universidad ORT Uruguay
Montevideo, Uruguay
florescia.raschetti@gmail.com

Abstract—Software development requires professionals with knowledge and experience on many different methodologies, tools, and techniques. However, the so-called soft skills, such as interpersonal skills, teamwork, problem solving and customer orientation to name just a few, are as important as, or even more important than, traditional qualifications and technical skills. Members of scrum teams, particularly the ones performing the roles of Product Owner and Scrum Master, are not exempt of having these kind of skills because of the distinctive duties and responsibilities of these roles in a Scrum team. In this paper we report a field study in which we interviewed 25 experienced Scrum practitioners from software companies in Uruguay to know their points of view about what are the soft skills they consider the most valued to have by the Product Owner and the Scrum Master of a Scrum team. As a result, Communication skills, Customer orientation, and Teamwork appear as the most valued soft skills Product Owner should have, while Commitment, Communication skills, Interpersonal skills, Planning skills, and Teamwork are considered the most valued ones for the Scrum Master.

Keywords- *soft skills; scrum; product owner; scrum master*

I. INTRODUCTION

Software development is a highly technical activity that requires people performing diverse roles in software projects, and with knowledge and experience on many different methodologies, tools, and techniques.

However, as people in software projects have to work together in order to achieve project goals, other kind of skills and abilities are also needed, related to the execution of project tasks such as interacting and communicating with teammates and stakeholders, managing time, negotiating with customers, writing reports, presenting project advances, problem solving, and decision making, among others alike.

These skills are examples of a broad compendium of several components like attitude, abilities, habits and practices that are combined adeptly to maximize one's work effectiveness [1], and they are considered as important as, or even more important than, traditional qualifications and technical skills for personal and professional success [2].

This kind of skills are known in literature as "soft skills", "non-technical skills", "people skills", "social skills", "generic competencies", or "human factors".

According to Capretz, the human factor is a make-or-break issue that affects most software projects and thus, an understanding of these factors is important in the context of the practice of software engineering [3].

In a previous study [4], one of the authors identified 17 soft skills that are usually demanded by software companies in Uruguay when hiring new professionals to work in software projects.

During recent years, several software companies in Uruguay have been adopting agile methodologies, particularly Scrum, for managing their software development projects.

Agile software development is carried out through the collaboration between self-organizing, cross-functional teams. Thus, agile teams depend greatly on efficient communication, taking responsibility, initiative, time management, and leadership [5], examples of the above mentioned soft skills.

As explained in [6], Scrum development efforts consist of one or more Scrum teams, each made up of three roles: Product Owner, ScrumMaster, and the Development Team. Of these roles, in this paper we will concentrate on the two of them that are unique and distinctive: Product Owner and Scrum Master.

The data used in the previous study reported in [4] was collected from job ads published in a major national newspaper of Uruguay, and from the database maintained by the Graduate Office of Universidad ORT Uruguay, that receives jobs ads directly from software companies looking for new staff.

In this paper, our purpose is to deepen that previous study to have the "insider" voices of Scrum practitioners about what are the soft skills they consider most valued to have by Scrum team members. Specifically, we wanted to have the separate perspectives of product owners, scrum masters, and team members of Scrum teams about what are the soft skills they consider of most value to have by their teammates, being either the Product Owner or the Scrum Master.

The remainder of this article is organized as follows. In Section II we give an overview of the three Scrum roles. In Section III, and since software engineering research in Uruguay is scarce, we will give a brief overview of Uruguay and its software industry. In Section IV we present the research questions posed for this study, while in Section V we describe the data collection process followed. In Section VI we present the analysis of the collected data and we answer the research

questions. In Section VII we compare the points of view of product owners, scrum masters and team members of Scrum teams regarding the most valued soft skills to perform the distinctive roles of Product Owner and Scrum Master. Finally, in Section VIII we present our conclusions and further work.

II. SCRUM ROLES

Mainly based on [6], what follows is a brief description of the three roles defined in the Scrum framework:

A. Product owner

The product owner is the empowered central point of product leadership. He/she is the single authority responsible for deciding which features and functionality to build and the order in which to build them.

The product owner holds the product vision, and must understand the needs and priorities of the organizational stakeholders, the customers, and the users well enough to act as their voice. In this respect the product owner acts as a product manager, facilitating communication between the team and the stakeholders to ensure that the right solution is developed.

B. ScrumMaster

The ScrumMaster helps everyone involved understand and embrace the Scrum values, principles, and practices. He/she acts as a coach, providing process leadership and helping the Scrum team and the rest of the organization develop their own high performance, organization-specific Scrum approach.

As a facilitator, the ScrumMaster helps the team resolve issues and makes improvements to its use of Scrum, and is also responsible for protecting the team from outside interference and takes a leadership role in removing impediments that inhibit team productivity. He/she also facilitates regular team meetings to ensure that the team progress to its path to "done".

C. Development Team

Traditional software development approaches discuss various job types, such as architect, programmer, tester, database administrator, UI designer, and so on. Scrum defines the role of a development team, which is simply a diverse, cross-functional collection of these types of people who are responsible for designing, building, and testing the desired product.

III. URUGUAY AND ITS SOFTWARE INDUSTRY

With a population of 3.2 million people, Uruguay has positioned itself in recent years as a leading exporter of software in Latin America. In 2013, exports of software and related services reached 300 million dollars, and CEOs of leading companies expect to reach 1 billion dollars by 2020. The main foreign markets are the United States, Argentina, Brazil, Spain, and Canada. At present, there are about 250 companies that produce software, that employs about 4500 professional, and the unemployment rate in this industrial sector is almost zero.

IV. RESEARCH QUESTIONS

We posed two groups of research questions for this study, as depicted below:

- RQ: A: What are the most valued soft skills a *Product Owner* must have, from the point of view of:
 - A1: a Product Owner, A2: a Scrum Master, A3: a development team member
- RQ: B: What are the most valued soft skills a *Scrum Master* must have, from the point of view of:
 - B1: a Product Owner, B2: a Scrum Master, B3: a development team member

Table I shows the cross relationship between these six research questions about Product Owner and Scrum Master.

TABLE I. RELATIONSHIP OF RESEARCH QUESTIONS

	About...		
		<i>Product Owner</i>	<i>Scrum Master</i>
Point of view of...	Product Owner	A1	B1
	Scrum Master	A2	B2
	Team Member	A3	B3

V. DATA COLLECTION

To collect data for this study, we interviewed 25 software engineering practitioners with working experience in Scrum, from 8 software development companies in Uruguay. These 8 companies were selected from the set of companies that posted the job ads used in the previous study mentioned above. Of these companies, 6 declared to use Scrum as an agile methodology and the other two a hybrid of iterative and agile methodologies. Regarding the years in the Uruguayan market, the youngest company is 4 years old, while the older is 23 years old, with an average of 10 years. With respect to the quantity of employees directly involved in software engineering tasks, the smallest company has 5 people, and the biggest one has 390, with an average of 40 people.

As mentioned above, the software engineering professionals interviewed for this work have working experience in Scrum. Four of them have experience as a Product Owner, seven as a Scrum Master, and the other fourteen have experience only as a member of a Scrum team.

In Table II we show the interviewees' minimum, maximum, and average years of experience in performing their respective roles as part of Scrum teams.

TABLE II. INTERVIEWEES EXPERIENCE WITH SCRUM (YEARS)

Role	Min.	Max.	Avg.
Product Owner	1	2	1.5
Scrum Master	0.75	4.5	3.1
Team Member	0.5	4.5	2.9

During the interviews, we gave the interviewees the list of the soft skills identified in [4] along with a conceptual definition of each skill.

To answer the six research questions, we requested the interviewees to select from that list the soft skills that he/she considers the most valued to have by a Product Owner (A's questions) and by a Scrum Master (B's questions).

VI. DATA ANALYSIS

With the data obtained from the 25 interviewees, the answers to the research questions posed for this study are as follow:

A. The most valued soft skills a Product Owner must have.

To answer the research questions A1, A2, and A3, we asked separately to product owners, scrum masters and team members to select the soft skills considered most valued to perform the role of Product Owner.

From the perspective of the four product owners interviewed, the top five soft skills considered most valued to have by a Product Owner (RQ. A1) are shown in Table III.

TABLE III. TOP FIVE SOFT SKILLS FOR PO (PO'S POINTS OF VIEW)

Soft skills	Times selected	%
Communication skills	4	100
Customer orientation	4	100
Interpersonal skills	3	75
Teamwork	3	75
Analytic, problem-solving	2	50

From the perspective of the seven Scrum masters interviewed, the top five soft skills considered most valued to have by a Product Owner (RQ. A2) are shown in Table VI.

TABLE IV. TOP FIVE SOFT SKILLS FOR PO (SM'S POINT OF VIEW)

Soft skills	Times selected	%
Communication skills	7	100
Customer orientation	7	100
Planning skills	7	100
Teamwork	7	100
Commitment, responsibility	6	85.7

Finally, from the point of view of the 14 team members interviewed, the top five soft skills considered most valued to have by a Product Owner (RQ. A3) are shown in Table V.

TABLE V. TOP FIVE SOFT SKILLS FOR PO (TM'S POINTS OF VIEW)

Soft skills	Times selected	%
Communication skills	14	100.0
Commitment, responsibility	10	71.4
Teamwork	9	64.3
Customer orientation	8	57.1
Motivation	8	57.1

B. The most valued soft skills a ScrumMaster must have

To answer the research questions B1, B2, and B3, we asked separately to product owners, scrum masters and team members to select the soft skills considered most valued to perform the role of Scrum Master.

From the point of view of the four product owners interviewed, the top five soft skills considered most valued to have by a Scrum Master (RQ. B1) are shown in Table VI.

TABLE VI. TOP FIVE SOFT SKILLS FOR PO (SM'S POINTS OF VIEW)

Soft skills	Times selected	%
Communication skills	4	100
Interpersonal skills	4	100
Commitment, responsibility	3	75
Organizational skills	3	75
Planning skills	3	75

From the perspective of the seven Scrum masters interviewed, the top six soft skills considered most valued to have by a Scrum Master (RQ. B2) are shown in Table VII.

TABLE VII. TOP FIVE SOFT SKILLS FOR SM (PO'S POINTS OF VIEWS)

Soft skills	Times selected	%
Communication skills	7	100
Interpersonal skills	7	100
Motivation	7	100
Teamwork	7	100
Commitment, responsibility, Planning skills	6 (each one)	85.7

Finally, from the point of view of the 14 team members interviewed, the top five soft skills considered most valued to have by a Scrum Master (RQ. B3) are shown in Table VIII.

TABLE VIII. TOP FIVE SOFT SKILLS FOR SM (TM'S POINTS OF VIEWS)

Soft skills	Times selected	%
Communication skills	13	92.9
Interpersonal skills	12	85.7
Leadership	12	85.7
Commitment, responsibility	10	71.4
Planning skills	10	71.4

VII. COMPARING THE POINTS OF VIEW OF PO, SM AND TM

Based on the data shown in Table III to VIII, we found interesting to compare the points of view of product owners, scrum masters and team members with regard of the most valued soft skills for a Product Owner and for a Scrum Master.

In the next two sub-sections we show the results of these comparisons.

A. Comparing the points of view of product owners, scrum masters and team members about the most valued soft skills a Product Owner must have.

For this comparison, we put together the data shown in Table III (point of view of product owners, PO), Table IV (point of view of scrum masters, SM) and Table V (point of view of team members, TM).

TABLE IX. PRODUCT OWNER: POINTS OF VIEWS OF PO, SM AND TM

Soft skills	PO	SM	TM
Analytic, problem-solving	X		
Commitment, responsibility		X	X
Communication skills	X	X	X
Customer orientation	X	X	X
Interpersonal skills	X		
Motivation			X
Planning skills		X	
Teamwork	X	X	X

Table XI shows the eight soft skills that appears in those tables and, grayed, the ones that are in common from the three perspectives.

From this comparison results that Communication skills, Customer orientation, and Teamwork are the three soft skills that appear as the most valued for a Product Owner, from the perspectives of product owners, scrum masters and team members.

B. Comparing the points of view of product owners, scrum masters and team members about the most valued soft skills a ScrumMaster must have.

For this comparison, we put together the data shown in Table VI (point of view of product owners, PO), Table VII (point of view of scrum masters, SM) and Table VIII (point of view of team members, TM).

Table X shows the eight soft skills that appears in those tables and, grayed, the ones that are in common from the three perspectives.

TABLE X. SCRUMMASTER: POINTS OF VIEWS OF PO, SM AND TM

Soft skills	PO	SM	TM
Commitment, responsibility	X	X	X
Communication skills	X	X	X
Interpersonal skills	X	X	X
Leadership			X
Motivation		X	
Organizational skills	X		
Planning skills	X	X	X
Teamwork	X	X	X

From this comparison results that Commitment, responsibility, Communication skills, Interpersonal skills, Planning skills, and Teamwork are the five soft skills that appear as the most valued for a Scrum Master, from the perspectives of product owners, scrum masters and team members.

VIII. CONCLUSIONS AND FURTHER WORK

In this paper we reported a field study in which we interviewed 25 software engineering practitioners experienced in Scrum from 8 software companies in Uruguay to know their opinions about what are the soft skills they consider the most valued to have by the people performing the role of Product Owner or of Scrum Master in a Scrum development team.

Based on the data collected on those interviews, to perform the specific role of Product Owner of a Scrum development team, the point of view of product owners, scrum masters and team members are coincident in that Communication skills, Customer orientation, and Teamwork are the most valued soft skills for performing that role.

To perform the role of Scrum Master, the perspectives of the product owners, scrum masters and team members interviewed are coincident in that Commitment, responsibility, Communication skills, Interpersonal skills, Planning skills, and Teamwork are the most valued soft skills to perform this role.

Findings suggest that there are much more coincidences than discrepancies between the perspectives of product owners, scrum masters and team members regarding what are the most valued soft skills software engineering professionals should have to better perform those two specific and distinctive roles defined in the Scrum framework.

As a further work, we are now working with the companies involved in this study to investigate, among other things: a) what impact have these soft skills of product owners and scrum masters in their software projects outcomes, b) what soft skills are found less developed in their product owners and scrum masters, and what actions are the best to take in order to develop those skills to enhance software projects outcomes, c) whether there are other soft skills, beyond the ones used in this study, that are important to perform those roles.

REFERENCES

- [1] G. Ramesh and M. Ramesh, *THE ACE of soft skills. Attitude, communication and etiquette for success.* New Dehli: Dorling Kindersley, 2010.
- [2] E. Kumar and P. Sreehari, *Communication skills and soft skills. An integrated approach.* New Dehli: Dorling Kindersley, 2011.
- [3] L. F. Capretz, "Bringing the Human Factor to Software Engineering," *IEEE Softw.*, vol. 31, no. 2, pp. 102–104, 2014.
- [4] G. Maturro, "Soft skills in software engineering: A study of its demand by software companies in Uruguay," in *6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2013, pp. 133–136.
- [5] L. Bender, G. Walia, F. Fagerholm, M. Pagels, K. Nygard, and J. Münch, "Measurement of the Non-Technical Skills of Software Professionals: An Empirical Investigation," in *26th International Conference on Software Engineering & Knowledge Engineering (SEKE 2014)*, 2014, pp. 478–483.
- [6] K. S. Rubin, *Essential Scrum: a practical guide to the most popular agile process.* Upper Saddle River, New Jersey: Pearson, 2013.

Reflecting, adapting and learning in small software organizations: an action research approach

Suzana Cândido de Barros Sampaio, Marcelo L. M. Marinho, Alexandre J. H. de O. Luna, Hermano P. de Moura¹

¹Informatics Center – Cin, Federal University of Pernambuco
Recife, PE, Brazil
{scbs2, mlmm, ajhol, hermano}@cin.ufpe.br

Abstract— Software Engineering activities are context based and carried out by people, within its culture and project actuality. Consequently, it demands a great deal of social relations. In order to better understanding these challenges faced by software development projects, we have had to go beyond the actual mindset, literature and bodies of knowledge. This paper is a result of an empirical research, aligned with evidence-based Software Engineering, about studies conducted on five software development Micro and Small Enterprises in Brazil, during 22 months, between July-2012 and May-2014. We have adopted a participant observer ethnographic study, resulting in intervention based on action research. The interventions happened several times, leading into continuous and constructive process of reflecting and learning. As a result, we have observed the emergence of a practical problem solving culture, from a collaborative immediate situation, which expanding the actor's competencies in every cycle of its execution. Although every organization had its own major problem to be dealt with, our findings point out to some common problems and emerging action strategies to handle with these challenges.

Keywords— *action research, qualitative research, project management, project actuality.*

I. INTRODUCTION

Over the past 20 years, there has been a substantial improvement in the quality and rigor of research in PM [1]. Project has become more relevant to organizational change and growth as it is used to achieve business objectives. According to Shenhar Dvir, Levy and Maltz, projects are a unique way for organizational change, innovation and face the competitive market's reality [2]. Mintzberg [3] advocate that Project Management (PM) is important to anyone who is effected by its practice that means the entire organizational world. In order to take full advantage from PM practice, organizations, teams and practitioners must adopt new ways of learning, thinking and reasoning in action.

Nowadays, research empirical method has become part of the Software Engineering research practice. In the experimental software engineering paradigm, the relationship between practitioners and researchers is highly symbiotic, where researchers need laboratories to observe and manipulate variables in vivo, and *project context* seems as an ideal environment to do that. In the other hand, practitioners feel the need to understand how best they can build and maintain their organizational system, and researchers can help them to

achieve this end [4]. In order to be more competitive, PM practitioner must also understand how to step up and improve its competencies, processes, overcoming their problems and achieving better results. Indeed, action research has emerged as a good strategy to accomplish that goal.

In addition, Micro and Small Enterprises (MSEs) have a great importance in any country's socioeconomic scenario, and in Brazil it is no different. A research conducted by the Brazilian Institute of Geography and Statistics (IBGE) in 2010 depicted that this kind of organization represents more than 99% (5.7 million) of Brazilian companies and they represent 60% of the jobs across the country. However, it represents no more than 20% (US 700 billion) of the Brazilian GDP¹ [5]. In fact, it is an expressionless percentage when compared to other nations, showing that there is still much opportunity for growth to this economic sector. Among ICT companies over 85% can be classified as Micro or Small Business [6].

From this perspective, and aiming at addressing this growth opportunity, as well as the expected benefits of organization and PM practices enhancement, we have conducted five action research studies in small software development organization in the northeast of Brazil. It is important to point out that the organizations in these studies did not know their real problems, that is way ethnographic techniques [7] were necessary in order to diagnosis the major challenges. Each organization, team or project had its own problems to solve. This is a part of a broader research [8] on project actuality were exploratory and systematic literature review [9] and ethnographic studies were conducted before the action research. By facing project as our research field and by involving the practitioners, with the real and major problems identified, researchers and practitioners can embrace the reflective practitioner in its actuality, willing to rethink, undo, redo and learn.

This paper is organized as follows. Next section gives an overview of the action research method, its main concepts, steps and principles. Section 3 exhibit a brief sample's summary, and five organizational contexts. Section 4 presents the research design and steps executed before the study. Section 5 describes the approach used. In addition, Section 6 presents some actions and feedback obtained as studies result.

¹ Gross Domestic Product.

Finally, in Section 7 final considerations and limitations are discussed.

II. ACTION RESEARCH

Scientific methodology is necessary to make the research results more reliable and reproducible by other researchers. According to Zerkowicz [10], the social challenges dealt with by researchers in Software Engineering investigations makes Action Research (AR) a useful research methodology due to its characteristics and possibility of obtaining relevant results.

While most empirical research methods attempt to observe the world, as it currently exists, action researchers aim to intervene in the studied situations for the explicit purpose of improving the situation [11]. The knowledge gathered from research empowers particular individuals or groups, and facilitate a wider change. The AR's application focus involves solving organizational problems through interventions, while at the same time contributes to teams and organizational knowledge. In our studies, we had the purpose of improving the organizations and its teams, by overcoming their problems and helping team members to engage in reflection.

Davison, Martinsons and Kock [12], suggests a unidirectional flow for AR, with diagnosis followed by planning, intervention, evaluation and reflection. In order to better suit to small enterprise and for teams' reflection, rethinking and learning, the major steps were adapted. In addition, to avoid just focusing on the iceberg top or in a specific effect (but not in its causes), we carried out the AR after ethnographic based study. Using ethnography as a technique to proceed the diagnosis stage from action research.

III. THE IN-VIVO SAMPLE

The research sample is comprised by MSE in the northeast of Brazil. Four of them were from Recife (A, B, C and D), and two of them were from a small city over 700km away (F and G). Our research is context and time dependent, and was conducted along from July, 2012 to May, 2014. **Organization A** has nearly 10 years of experience in the IT industry, founded in 2004 to provide solutions in managing industries, services and trade. They have a product developed in Delphi with firebird and one in Java JSP for another purpose. A team of five developers took care of over ten clients. The owner was centralizing and the ambient was noise and the Delphi team was demotivated and tired. The room and machines were new, clean and organized. The most expected problem to be solved was to overcome not being able to estimate correctly and precisely.

Organization B was the oldest and biggest one in the sample with 25 years in the market, and two teams. The organization had two solid products, with the biggest clients, also partnership with other organizations that combined presented an even more solid and wider solution. If a client wants something different but still related to their products, they would get the challenge to gain market. Both observed teams worked in two distinct Delphi product. The biggest challenge for their teams was to stop clients and manager interference that generated lack of commitment and

motivation. In addition, the major challenge for the organization was to overcome lack of visibility and trust on teams work. The organization had level G of MPS-SW [13] and already had their work organized as projects.

Organization D was the most mature one, used to have MPS-SW [13] level G, but it expired. With 13 years in the market, it had a nice renewed office. The owner was from IT world, but along our research, we never saw him with the team. It was a family business, although the manager was not part of the family. The study initiated with one manager and changed along the way. The change was freighting, but team got more cohesive, less bureaucratic and productive after this change and along our study. Besides de developers, one tester, quality assurance that could also develop made the team. A part time trainee tester was also involved along the project.

Organization F was the hardest one to commit to the research. They had the nosiest room, the worst scenario and took longer to build trust and finally engage in action. The owner was really busy selling, but he originally messed everyone's plans. He gave a percentage in the society for his two best men, both with over 15 years working with him. With 2 hours lunchtime, 100% of the organization went home to eat and rest. The either walked home or used motorcycle to get around. Many times the first researcher waited outside in over 35° degree Celsius, feeling 45° sun.

Out of them all, **Organization G** had the most technical PM, one of the two owners. One owner dealt with sales, and took care of the support team. The other, great researcher in action and enthusiast of our research. He manage to learn quick, try hard to rethink, reflect and move forward. He counted as one developer, although his help was sporadically. One developer was 30 hours only, but one of the best programmer of the team. With two hours lunch, everyone went home. A young nice and committed team.

IV. PREVIOUS TO THE ACTION

A. Planning the studies

The organizations were chosen by convenience, all of them were indicated by SoftexRecife². It supports micro small software development organizations through training, process improvement, testing and other services.

The authorization was written as an invitation letter-consent form signed by the organization high management, or site coordinator and the first researcher. The letter states researchers' names and affiliation, the research goal, procedures and techniques and confidentiality rules. The end criteria was a bit abstract, but the idea was to keep on researching until the main problems are overcome, challenges are dealt with and both parties involved are satisfied. In addition, a research plan was presented to the team to guarantee their understanding and commitment.

² The Software Technology Excellence Center in Recife, a civil non-profit association, established on Nov.8th, 1994 with a mission to increase the competitiveness of ICT companies (<http://www.recife.softex.br/>).

B. Diagnosis – The Ethnographic Study

Schensul [14] presents ethnography as a scientific approach to discovering and investigating social and cultural patterns and meaning in communities, institutions, and other social settings. Ethnographic studies in software engineering are valuable for discovering what really goes on in particular technical communities, and for revealing subtle but important aspects of work practices [11]. We saw as way of unveiling the major problems, faced by the team or project manager independent of the source. All problems were grouped in a backlog, prioritized by the manager, and always revised along the intervention. Every finding came from several evidences and a causality analysis made in the ethnographic study.

Table I depicts the most common problems in between the five organizations. The **problem backlog** was presented in questions in order to confirm it and prioritize them. Moreover, the idea was also empower the reflexive practitioner and the practitioner-researcher. To each organization was also pointed the evidences that pointed or corroborated each problem.

TABLE I. MOST COMMON PROBLEMS

Org.	Problem Backlog	
	Problem	Reflecting question
A, B, F and G	(I) Absence of Reflection	Does your team have any time to reflect about what went wrong and what could have been done better?
A, B and F	(II) Blaming Culture	Is the blaming mood around? Is your team more worried about blaming someone for the bug, problem or issue than to solve it?
A, B, D, F and G	(III) Blind Capacity	Do you know what your team is capable of? How much work can you do, how much requirements can we compromise in a week, a cycle, a month?
A, B, D, F and G	(IV) Living the Problems	Are living the problems instead of solving the problems?
A, B and F	(V) Unproductive environment	Is your team ambient too noisy or the interruptions are driving away your productivity? What are nowadays our team's productivity villains?
A, B, F and G	(VI) Unclear Goals	Do you have an unclear goal? Can't your team know or tell what must be done?
A, B, F and G	(VII) Lack of Quality	Is your product producing more bugs then you can correct? Is your backlog of bugs bigger then you requirements backlog?
A, B, F and G	(VIII) Lack of Visibility	Are you unaware of what is going on with your team's activity?
A, F and G	(IX) Endless Operation	Is your work with no end neither beginning and seems to take forever with no partials winnings?
A, B, F and G	(X) Lack of Commitment and trust	Are all your initiatives top down? Is the owner or team leader the only one doing the talk? Is the goal always unrealistic? Do you care to make sure that your team buys what you say?
A, B and F	(XI) Lack of Motivation	Does your team never catch a break? Do you remember that they are people? Does your team's fear slow them down? Cannot see what can you do next?

V. THE APPROACH

This section presents the approach that encompasses the action research, team reflection and learning. Beginning with the adaptation from the Cyclical AR Process Model [12] followed by the reflection process.

A. Cyclical Research Action Process Model Adaptation

The *Diagnosis* was substituted by the research setup and the ethnographic study, as explained last section. In this stage the research field was defined, an agreement is settled and the problems are analyzed by the ethnographic study, similar to what Davison, Martinsons and Kock [12] proposal. The *Action Planning* stage uses the same idea as the one presented by the authors, where actions are defined for the diagnosed problems, although we included a reprioritization in each cycle, accommodating new prioritized-disturbing problems.

Intervention corresponds to the planned actions implementation exactly as idealized by the author [12]. The biggest adaptation is surrounding the *Reflection* activity; it does not only support the information flow between participants and the organization as presented by Davison, Martinsons and Kock [12]. As we deal with small organization, most organization's member were involved as researchers in action and just organization B needed to make the information flow, all the others had all team members engaged and involved. In our studies, reflections occurred before *Action Planning*, as presented in Fig. 1.

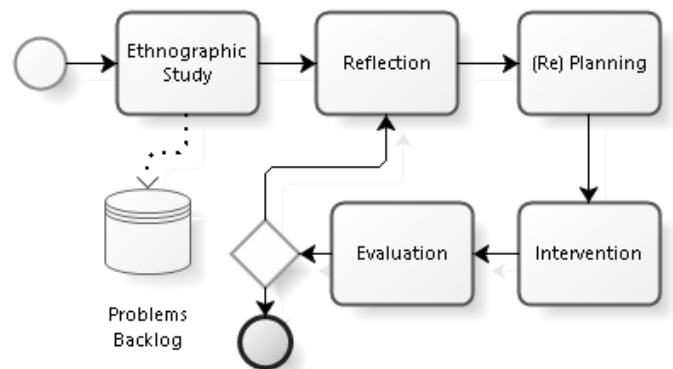


Fig. 1. Action Research Strategy. Source: Own elaboration.

After the intervention, the *Evaluation* took place, following the authors' idea of using theoretical support to analyze actions' effects and results. It was followed by another cycle started by reflection and learning activity.

B. Reflection Activity

The reflection sections represented a learning trigger to each context or problem discussed. It allowed the team to familiarize with some theory and question their status quo, in order to better see the problem and engage in overcoming it. The reflection activity was carried with the help of a macro activity and has the following goals: (i) conduct active interview or participant observation [15], [16]; (ii) get those who are being researched to play an active role in the process; rather the being passive subjects [16]; and help team to engage

in reflection about the best way to overcome a problem, a conflict or an improvement opportunity.

Every time an opportunity is identified, is a good time to reflect, although we used the reflection to start the cycle, to analyze the problems, situations and to rethink some ideas, approaches and strategies, resulting in the action plan. The steps necessary to accomplish this activity were:

- Identify conflicts, problems and singularities;
- Consider project actors researchers in action who must continuously question their actions and intentions in light of real-world situations [17];
- Evolve the reflection around questions [16], such as How, When and Why;
- Talk about the identified problem, conflict, singularities. Ask how can they overcome such problem, or how could it be done in a different way;
- Stimulate reflection; invite them to challenge the status quo, to analyze different ways of reasoning. And;
- Document findings. Always take notes, the quote spoken, the actor involved, the situation it came out and context it occurred.

As these activities' result, we have identified actions from project reflection, and an action plan was created.

VI. ACTIONS

Plenty where the actions we carried along the study. Some of them target not just the problems but foundation for future actions. For example, problem *I*, Absence of Reflection, without overcoming this barrier, probably once the study finished they would stop the reflection and rethinking. For that, one of the first actions was to introduce a Retrospective Meeting to all the organizations in the sample, except for *D* that already used it. Each organization organized itself differently as presented in Table II.

TABLE II. REFLECTION MOMENT.

Org.	Reflection Moment chosen by each Organization
A	Meeting after each important deliverable - end of implementation cycle.
B	Retrospective meeting after the sprint - two or three weeks sprint.
D	Meeting after two sprints of a weeklong.
F	Meeting every milestone, monthly.
G	Retrospective meeting in the end of every other sprint.

Another problem that we have faced was the Endless Operation (IX). Only **Organizations B** and **D** were already organized as projects. The problems effects were many. One of the evidences for **Organization G** was an affirmation in a meeting saying, "Today there are about four versions per month or more. It is costly and takes too much time." Every organization reflected about their own project definition, such

as: new product version; a product gap to be accomplish in order to meet customer needs; a slice of time, two to four weeks of work, with demands from several clients or several systems; a new system module delivery to meet legal demand; among others more traditional. Even though **Organization B** was already organized as projects, the reflection regarding the project concept was necessary due another problem, blind capacity, the difficulty on coordinating outside project with inside projects. The reflection included the higher management, the operational director and some other leaders that were senior employees. They were only organized timely (every 6 weeks a new official deploy), and the objective was to synchronize with outside projects and the main strategic goals. For that every two weeks they had their own "portfolio" meeting and they initiated to prioritize demands that would turn into projects. A few "outside small projects" (gaps to implant a product in a new client, legal demands, new demands from a strategic client) could turn into one project. In addition, a medium "outside project" (new product) could turn into a few projects. For the higher management this was the best result from our research.

In response to problem VII, all organizations in the sample reflected about the quality of their products and the return rate (associated with number of requirements with bugs). Different actions were conducted, although almost every team manage to initiate or enhance their testing tasks and skills. Rethinking the activities already done about the product's quality, some organizations figure it out that they had no "done" concept for their tasks and no success definition for their cycle or project. **Organization B** found problems such as: "Deployment on the client currently generates inconvenience to the elaborated Project Plan. Several development and testing activities are carried out in the field and sent to the team impacting business goal and the quality of the issued release"; and "team member's activity were submitted on branch of the Project, authorized by the PO, impacting activities and quality of the product and project". Some actions tried to address this misuse of the process and practice.

In resume, the organizations engage on several cycles of rethinking in order to "Enhance the product's quality", with the following derivate actions related to test: new test activities (A, F, G); work with product risk analysis (A, F); using support analyst for testing (A, F, G); hiring or allocating new test force (B, D); acquiring New tools – Testlink³ (D).

Problem V, related to lack of productivity, had the most unexpected and different actions. One of them was called "Major changes in order to achieve productivity or cohesion teams". As an example, **Organization B** had a re-distribution of the teams inside their room to sit together, reduce the noise and facilitate the communication. **Organization G** had a total change in the organization's physical structure. Coming from a single room with a hybrid profile (developer + support analyst) to everyone, but the owner, to separate rooms and separate positions. After four years, the development team got its own room and had only developing activities with three people only, plus two open spaces. The senior developer, that is also a small partner, took over the team management. One

³ <http://testlink.org/>

developer stayed in the support room, the only one really divided in between teams (support and developer), but he came to the room as demanded. The support team's room was divided with the owner, by a full brick-wall and some glass. A huge task board was organized. In addition, 10 minutes break every four hours to do whatever they want; better use of inside phones and less yelling around, as well as a mobile phone politics for a better-focused and productive group. In general, the action was the definition of a good coexistence policy or productivity policies, towards a productive team.

Many actions were executed facing the problems exhibit in Table I. Always coming from a situation or problem and aiming on overcoming it, such as: going from not knowing what we are capable of and how much time is necessary to accomplish some task, to estimative using complexity and relative sizing; from living the problems to solving the problems; from lack of behavior competence to better leadership skills; from lack of visibility to a transparent management system; and so on. Theories were used to present during reflections section as possible strategies and techniques.

Each organization took at least three formal cycles (D and F) to six cycles (B). In each cycle, different problems were faced and dealt with. Feedback from the AR, was qualitative. We recorded the feelings of progress and evolution in all problems reflected and treated. The most interesting ones were:

- **B** – *“The Portfolio vision was the best action of all. We can finally be predictable and we can finally give some pre-visibility to our clients”*.
- **D** – *“We had tried that once by ourselves and did not work. We got more agile and threw away heavy-casted process. You were our fairy godmothers”*.
- **G** – *“Every time you came here we learned something new or we gain some new perspective. We are more organized and predictable now; we will try to keep the reflection at least every other month”*.

VII. CONCLUSION AND FUTURE RESEARCH STUDIES

The idea of this study from the beginning was to help small software development organizations to achieve better results, stay competitive, enhance teams' competences and do not let projects be predestined to fail or to lack of its potential. In order to go beyond the “out of the shelf solutions”, action research helps to empowered reflexive practitioners to rethink the status quo and overcome their problems. We encourage a management thinking that inspires different ways of reasoning, reflecting and learning.

This paper presented the action research approach used as a reflecting, adapting and learning tool in small software development organizations in the northeast of Brazil. This step was primordial to a larger research within the experimental Software Engineering; aiming on understanding project actuality, and how to support small organizations to engage in reflection and learning even after the researchers have left the

research field. Our findings denote that Software Engineering is all about reflecting and learning as a team.

This study empowered a few organizations and teams to work as reflexive practitioners. Great changes happened after a few cycles of action research. Although this research is context dependent, we sure believe that still leaves a great opportunity for further work to improve small and medium software development organizations, this large potential market in Brazil.

ACKNOWLEDGMENT

This study would not have been possible without all the support and believe from the five organizations that opened their doors to us and engage themselves as researchers in action, willing to listen to us and reflect. The authors also would like to thank CAPES for supporting this research. And to SoftexRecife for introducing us to the organizations in the sample.

REFERENCES

- [1] J. R. Turner (2010). Evolution of project management research as evidenced by papers published in the International Journal of Project Management, 28: 1-6.
- [2] A. J. Shenhar, D. Dvir, O. Levy and A. C. Maltz (2001). Project success: a multidimensional strategic concept. Long range planning, 34(6): 699–725.
- [3] H. Mintzberg (2009). Managing. Berrett-Koehler Publishers.
- [4] V. Basili, R. Selby and D. Hutchens (1986). Experimentation in software engineering. IEEE Transactions on, (7), 733-743.
- [5] IBGE (2010). Reserach on Comunication and Information Techonologies in organizations, 2010. [ftp://ftp.ibge.gov.br/Tecnologias de Informacao e Comunicacao nas_Empresas/2010/comentarios.pdf](ftp://ftp.ibge.gov.br/Tecnologias_de_Informacao_e_Comunicacao_nas_Empresas/2010/comentarios.pdf) accessed in Ago 8th 2014.
- [6] ABES, 2013. Brazilian Software Market: scenario and trends. 1Ed. São Paulo: ABES.
- [7] M. Hammersley and P. Atkinson (2007). Ethnography: Principles in practice. Routledge.
- [8] S. C. B. Sampaio, M. L. M. Marinho and H. P. Moura (2014). An Approach to Understanding Project Actuality in Small Software Development Organizations and Contribute to Their Success. ProjMAN - International Conference on Project MANagement, 2014, Troia.
- [9] S. C. B. Sampaio, M. L. M. Marinho and H. P. Moura (2014). Systematic Review on Project Actuality. IJCSIT.
- [10] M.V. Zelkowitz, D. Wallace (1997). Experimental validation in software engineering. Information and Software Technology, 39 (11), 735-743.
- [11] S. Easterbrook, J. Singer, M. Storey, D. Damian (2008). Selecting empirical methods for software engineering research. Guide to advanced empirical software engineering, 285–311.
- [12] R. Davison, M. G. Martinsons, N. Kock (2004). Principles of canonical action research. Information systems journal, 14(1), 65-86.
- [13] SOFTEX (2012). MPS.BR General Guide. Mps.br guia geral 2012.
- [14] Schensul, S. L. (1999). Essential ethnographic methods: Observations, interviews, and questionnaires (Vol. 2). Rowman Altamira.
- [15] S. Cicmil, T. Williams, J. Thomas and D. Hodgson (2006). Rethinking project management: researching the actuality of projects. International Journal of Project Management, 24(8), 675-686.
- [16] S. Cicmil (2006). Understanding project management practice through interpretative and critical research perspectives. Project management journal, 37(2), 27-37.
- [17] L. Crawford. Developing organizational project management capability: theory and practice. Project Management Journal, 37(3):74–97, 2006.

Application of Slow Intelligence Framework for Smart Pet Care System Design

Shi-Kuo Chang¹, Wen-Hui Chen², Wen-Chyi Lin³ and Christopher Lee Thomas¹

¹Department of Computer Science, University of Pittsburgh, USA ({schang, clt29}@pitt.edu)

²Graduate Institute of Automation Technology, National Taipei University of Technology, Taiwan (whchen@ntut.edu.tw)

³Department of Electrical and Computer Engineering, University of Pittsburgh, USA (wel69@pitt.edu)

Abstract—This article presents the design of a smart pet care system based on the slow intelligence framework for providing pets with suitable living conditions that closely mirror their natural habitat. By integrating heterogeneous information from various sensing data, the smart environment-aware pet care system can adaptively adjust the setting of temperature and humidity that best fits for the pet through iterative slow intelligence computation. Simulations of two case studies were provided to illustrate the application of the proposed system for pets such as snakes and dogs. The simulation results demonstrate the feasibility of the proposed approach to the design of smart pet care systems.

Keywords- Smart pet care systems; slow intelligence systems; environment-aware software engineering

I. INTRODUCTION

Unlike human health care systems, pet care systems require more autonomous mechanisms to perceive and respond to the changes of environmental conditions as pets are unable to alert their caretakers if an anomalous condition arises. As pets have their specific living conditions to thrive, a pet care system for one species may not be suitable for another. Therefore, component reuse software architecture is beneficial for the design of a pet care system.

Software reusability is essential and plays an important role in building a scalable system from software component reuse and integration. In this study, we proposed a component-based software framework based upon slow intelligence systems to integrate existing software components into self-regulating and adaptive systems. A slow intelligence system (SIS) is a general-purpose system characterized by being able to improve its performance over time [1]. An SIS is characterized by employing super-components, in the sense that multiple components can be activated either sequentially or in parallel to search for solutions. We have developed a visual specification approach using dual visual representations, and the user interface to design a component-based system based upon the dual visual representations [2].

The proposed software framework emphasizes the design of reusable software components so that the developed system can be easily applied to various pets. To implement an environment-aware system for pet care, an adaptive control strategy that adopted the principles of slow intelligence systems was proposed. The proposed environment-aware system can sense and react on the environment accordingly to

provide pets with a proper temperature and humidity setting through continuous interaction with the environment by evolutionary computation that carries out adaptive control operations.

The caring system for a pet snake is different from the one for a pet dog or a pet cat. To validate the proposed system is appropriate for various pets, we used the dog care and the snake care as two illustrative examples to demonstrate the feasibility of the proposed framework for the application of a smart pet care system.

The remainder of this article is organized as follows. Section 2 introduces the framework of the proposed pet care system. Section 3 describes the application of the proposed framework for monitoring pet snakes and pet dogs. Section 4 provides simulation results and discussions. Conclusions are drawn in Section 5.

II. THE FRAMEWORK OF SIS-BASED PET CARE SYSTEM

Considering the software usability and scalability, we proposed a component-based software architecture for pet care system design. The proposed framework, as shown in Fig. 1, consists of eight software components, including Universal Interface, SIS Server, System Interface, Alerter, Pet Activities Monitor, Environment Sensor, Image Sensor, and Image Processor.

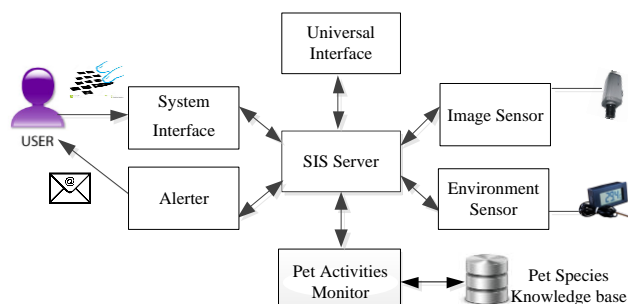


Fig. 1. The software architecture of the proposed pet care system.

2.1 Slow Intelligence System (SIS) Server

The SIS server is responsible for routing and processing messages among components. When the SIS server accepts a request message from a component, the message will be processed and routed to the designated components by the SIS server as responses. All request and response messages are

encoded in the XML format for usability across the Internet. Some particular messages were defined for the SIS server to execute specific operations. For example, messages Create, Kill, and Activate are used to enable the SIS server to create, kill and activate a user defined component, respectively.

2.2 System Interface (SI)

The System Interface component provides a graphic user interface (GUI) for users to specify system properties, including pet names, which e-mail address an alert should be sent to, the thresholds used by Image Processor, and the parameters required for reasoning algorithms. Note that this component does not receive any messages from the SIS server, and has no control logic inside it. It is only a GUI for users to specify system properties. The properties set by this component are stored in a shared database for other components to access. Thus, there is no need to design extra communication between this component and other components. If the user does not set the properties, default properties will be used.

2.3 Universal Interface (UI)

Universal Interface is used to simulate routing messages among components. It is useful when testing the developed system by observing routing messages. The user can enter a message on the left panel, and observe the corresponding output message on the right panel. Any message sent from one component to another component will be displayed on the right panel.

2.4 Pet Activities Monitor (PAM)

Pet Activities Monitor is designed to recognize pet activities by analyzing sensor data from Environment Sensor and Image Sensor. Different types of sensors require different data processing. PAM consists of four units: a data preparation unit, an image processing unit, an image recognition unit, and a decision unit. The data preparation unit is to convert sensing data to feature vectors. The image processing unit is to process images captured from Image Sensor and perform feature extraction for the image recognition unit, while the image recognition unit is to recognize pet activities. The decision unit is to make a decision of whether an alert needs to be sent to the caretaker about an anomalous condition.

When an MSG 33 message is received by PAM, the image processing unit will start to load the image captured from Image Sensor and its timestamp described in MSG 33, and perform specific activity recognition algorithms.

2.5 Alerter

The Alerter component is responsible for receiving an MSG 38 message and sending an alert e-mail or text message to the user to notify an anomaly is detected so that the user can take action accordingly.

2.6 Environment Sensors (ES)

The Environment Sensors component is designed to read and store all available sensor data, such as temperatures and humidity for observing environmental conditions. In addition

to analog sensor readings, Environment Sensor also can handle binary sensor data such as the status of limit switches.

2.7 Image Sensor (IS)

The Image Sensor component is designed to read and store the image frames captured from the external camera. The location where the camera is installed depends on the purpose of applications. In the case of pet snake care, the camera could be mounted atop the water bowl, while in the case of dog care, the camera could be placed in a location where the dog can be easily observed. When a new captured image is generated, a message MSG 33 will be generated and sent to notify the SIS server. The stored images are analyzed by the image processing unit in Pet Activities Monitor. As the observed pet may not sit near the server or may not even be in the same building as the place where the pet lives, it is necessary for Image Sensor to support different camera inputs, such as IP cams and webcams for remote access.

III. APPLICATIONS OF THE PROPOSED FRAMEWORK FOR PET CARE SYSTEMS

3.1 Description of Pet Snake Care

Snakes are poikilothermic animals and require environmental heat for various bodily activities. Although commercially available snake Vivaria can provide a housing enclosure with heating, water bowl, and covering, most of them lack of sensing and adaptive temperature control mechanisms for snake habitat monitoring, which can lead to a severe injury to pet snakes when the heater malfunctions.

The ideal habitat for pet snakes is not only monitoring the temperature on the hot side or the under tank heater to avoid a thermal burn, but also on the cool side to prevent the temperature from dropping too low. In addition, the humidity of the environment must also be monitored and kept in a proper setting. Even with these constraints, pet snakes still need the temperature lowered or raised according to their activities at all times, such as during shedding, after eating, etc.

The activity of a pet snake reveals some information about its status. For example, if it goes into the water bowl, it usually indicates the snake either wants to cool off or it feels sick. Although the information of living environment can be obtained by acquiring data from environment sensors, it is insufficient to adjust the temperature and humidity properly merely according to the sensor readings. As the proper setting may vary depending on their activities. In order to observe basic activities of a pet snake, an IP camera atop the water bowl is used.

A pet snake needs a suitable enclosure and some essential equipment inside the enclosure, like under tank heaters, water bowls and environmental sensors such as temperature and humidity sensors. The size and material of the enclosure may vary from one snake species to another. So does the equipment inside the enclosure. For example, some small snakes become anxious when living in a big space. The selection of the hardware equipment for pet snakes needs domain experts and beyond the scope of this research. Therefore, we assume the hardware enclosure as

well as the required equipment have already properly decided and installed, and only focus on the design of the software system for a smart pet care system.

3.2 Pet Activities Monitor for Snake Care

The Pet Activities Monitor component for pet snake care contains the fuzzy inference engine and fuzzy rule base for making a decision under uncertainty. The fuzzy decision starts with sensor data reading from environment sensors and goes through the fuzzification process. The inference engine works with attribute values that have fuzzy memberships defined, and produces a fuzzy output using max-min of inference. The output values are then de-fuzzified to a crisp value by the use of centroid de-fuzzification. The fuzzy rules and the parameters of membership functions are specified by users through the SI component.

Image Sensor provide the information of whether or not the observed snake is drinking out of the water bowl or lays in the water bowl by analyzing the captured image through vision-based scene analysis. In this experiment, we used the speeded up robust features (SURF) algorithm as an example for snake head detection to detect whether the head or the body of the snake is present in the water bowl and specified a value between 0 and 1 for these two possibilities.

The training images were collected and manually annotated for SURF feature descriptors. The SURF descriptors are feature vectors describing certain sections of an image. The image processing unit first computes the SURF feature descriptors over all the training images for both drinking and soaking. Fig. 2 shows two sample images with SURF feature descriptors for a Python Regius drinking and soaking in a water bowl. This is done to avoid re-computation of these values for each image received. The rationale is that images whose SURF descriptors match given thresholds are likely to be showing the same object.

When an MSG 33 message is received, the image processing unit will start to load the image and perform the SURF algorithm to detect whether the head or the body of the snake is present in the water bowl. If either of these are true, the probability of there being a problem is increased.

3.3 Iterative Slow Intelligence Computation

To determine an appropriate setting of temperature and humidity, we adopted iterative slow intelligence computation that contains four primitive phases, namely: enumeration, adaptation, elimination, and concentration defined in slow intelligence systems. A slow intelligence system is able to observe and act on the environment to achieve adaptive temperature and humidity control through the iterative computation process.

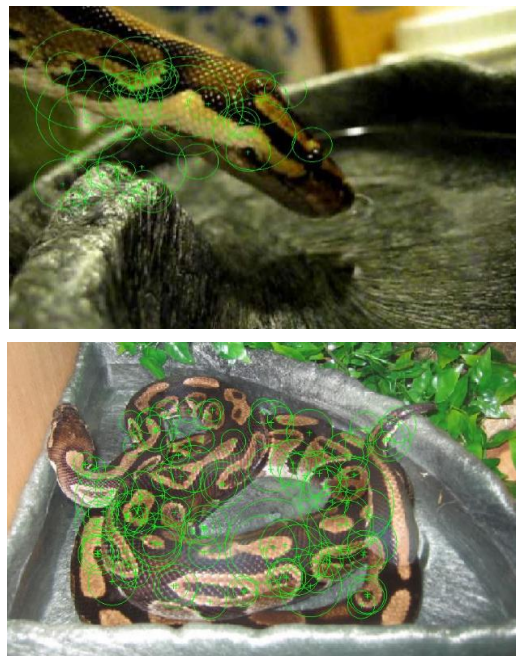


Fig. 2: Sample images with SURF feature descriptors for a Python Regius drinking and soaking in a water bowl.

In the proposed system, the possible combinations of temperature and humidity settings are considered as candidate control plans, which is the process of enumeration in slow intelligence system. At first, a set of fuzzy rules was built based on domain experts and was stored in the fuzzy rule base. Elimination is the process of ruling out un-matched SURF feature descriptors when comparing the query image with training images. This makes the recognition process more efficient as resources are only focused on matched descriptors, which is a formation of concentration. The proposed system is environment-aware by constantly interacting with the environment. The setting of the under tank heater could cause the changes of the temperature and humidity in the environment, so an update of the setting according to sensing data is required, which is the process of adaptation in slow intelligence computation. In Fig. 3, The SIS framework is shown where the circle and the timing control illustrates a super-component of an iterative slow intelligence system [3].

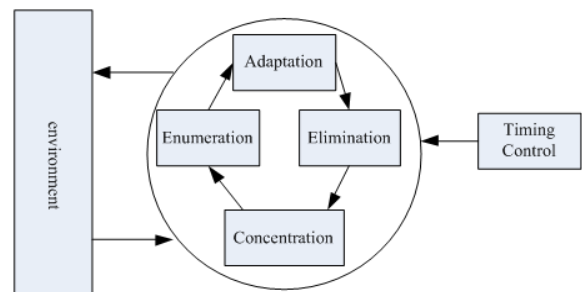


Fig. 3. The SIS framework where the circle and the timing control illustrates a super-component of an iterative slow intelligence system

The information obtained from Image Sensor is integrated with the information from Environment Sensor as data fusion so that the procedures performed by Pet Activities Monitor

can decide if an alert should be sent or not. The procedures performed by PAM are described as follows.

- Step 1: When a captured image is loaded, the SURF descriptors are computed on that image to identify regions of interest in the image.
- Step 2: The SURF descriptors are matched using the k -nearest neighbor algorithm against the pre-computed descriptors of every training image in the training set. In this study, we set $k = 1$ to find the closest matching descriptor and then compute the distance. That is, each vector associated with each descriptor in the captured image will be matched with one vector of the set on each training image. The matched vector is the one that has the least Euclidean distance.
- Step 3: For each of the matched sets, the overall normalized match strengths are computed. The cosine distance between each of the matched vectors was adopted as similarity measure.
- Step 4: Those matches whose distance is greater than the experimentally determined threshold are considered as bad matches and are discarded. Note that this step is critical as it will eliminate many bad matches, but also preserve the good matches as the process of concentration. The image containing the most matches is used and the other image matches are discarded.
- Step 5: The count of matches for the final image above the threshold value is input into a sigmoid function which translates the number of matches into a probability value to be used by the fuzzy inference engine inside the Pet Activities Monitor component.

3.4 Pet Activities Monitor for Dog Care

Living in an excessively cold or hot space can make a dog feel uncomfortable or even become life-threatening. Not all dogs are created equal. Different breeds of dogs have different hair coats. Breeds from cold climates usually have a downy coats and are much better at conserving heat than at cooling themselves. Therefore, the range of comfortable ambient temperatures for dogs to live varies from one breed to another. It is not the temperature but also the humidity that can affect dogs' health. If the humidity goes too high, dogs are unable to cool themselves, leading to their temperature raising to dangerous levels. As such, the best temperature and humidity settings in the dog housing environment should take dog breeds into consideration.

As the proposed system is designed in the framework of component reuse, only the PAM component in Fig. 1 needs to be modified for the design of a dog care system. Therefore, in this section, we only describe the dog breed recognition algorithm implemented in the PAM component. With this function, the proposed SIS-based pet care system can adaptively adjust the ambient temperature and humidity to a suitable setting for the pet dogs according to their breeds.

The knowledge required for proper temperature and humidity settings can be extracted and collected in the knowledge base through interviewing veterinarians and dog

experts. The PAM component comprises of several distinct classifiers and is integrated with the SIS Server to identify a dog breed from the camera. The information flow for dog breed recognition in the pet care system is shown in Fig. 4.

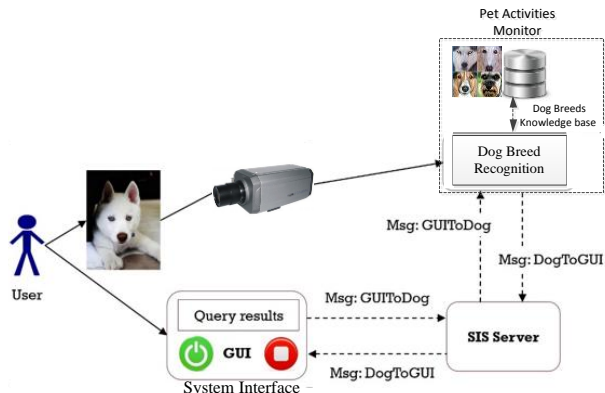


Fig. 4. The information flow of dog breed recognition in the pet care system.

The dog face detection and dog breed recognition algorithms are adapted from the Haar feature-based cascading classifier [4] and the FisherFace classifier, respectively. To process images rapidly and achieve high detection rates, the integral image is applied to compute the features used by the Haar feature detector. The Haar features are a set of directional filters that consist of different combination of rectangles. Based on the integral image, different sizes of Haar feature-based filters can be used to detect the region of interest (ROI) rapidly from the background in an image.

The sample dog images are adopted from the Stanford Dogs dataset [5] which contains images of 120 dog breeds. Fig. 5 shows four sample images and their cropped normalized images to a size of 70×70 pixels. This normalization is used to avoid extremely weights and maintain numerical stability.

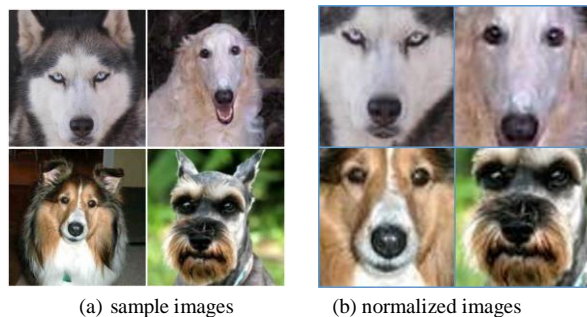


Fig. 5: Sample images from the Stanford Dogs dataset

Fig. 6 shows that the eyes and nose of a Borzoi dog are detected by two Haar feature filters and thus this sub-image can be accepted for further processing.

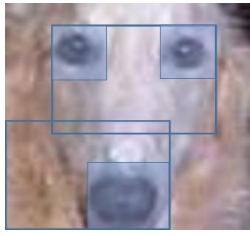


Fig. 6: Eyes and nose of a Borzoi dog are detected by Haar feature filters

As a particular case of ensemble learning, we generated a cascading classifier by combining increasingly more complex classifiers to allow background areas of the image to be quickly discarded while saving computing power on more dog face-like regions. As illustrated in Fig. 7, in the first stage of cascade, we remove the most unwanted backgrounds by using a coarse grained filter to favor speed. The last stage of cascade is a fine grained filter which can handle more detailed object features, then the detected result is passed to the FisherFace classifier for performing dog breed recognition.

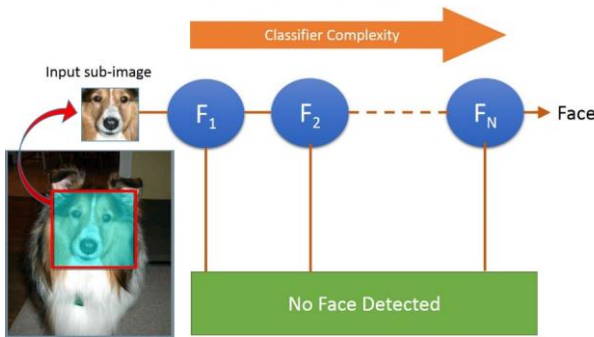


Fig. 7. Schematic diagram of the cascading classifier

Based on the Fisher's linear discriminant analysis, the FisherFace is proved to be robust against variation in lighting and facial expressions [6]. Fig. 8 shows three FisherFaces of the dog images in Fig. 5 (b).

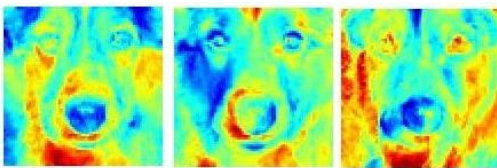


Fig. 8: Sample FisherFaces of dog images

IV. SIMULATION RESULTS

In this section, we describe the simulation results of the proposed approach by using the SIS testbed developed by the University of Pittsburgh. This testbed is a platform designed for developing SIS projects, and is available online at <http://people.cs.pitt.edu/~chang/163/interface/SequenceSIS.htm>

4.1 Simulation for Pet Snake Care

Before activating the system, some information needs to be specified as system properties through the SI component. SI provides a graphic interface for users to determine pet names,

alert e-mail address, the thresholds for Image Processor, and the parameters for the fuzzy inference engine.

The camera at the time of experiment was sitting on top of the snake enclosure pointed down towards the water bowl, and the Image Sensor component polls the camera every five seconds for a new image. Once the image is acquired from the camera, it is saved to a temporary location for further analysis. The message MSG 33 is broadcast to the SIS system to acknowledge that the image processing unit can begin processing the image.

The Environment Sensor component reads sensor data and broadcasts it to the system. In this experiment, we prepared a file with sensor readings as sensor data for simplicity. Environment Sensor repeatedly reads temperature data from the file to demonstrate the function of Environment Sensor. In the practical system, it would read from the real sensors.

The image processing unit begins processing images when it receives the message which contains the path to the captured image file from Image Sensor. The image has the keypoint descriptors calculated over it, which are then matched with the soaking and drinking data sets. After processing, it outputs two probabilities (in bowl probability and drinking probability) which are sent to Pet Activities Monitor.

The drinking probability and the in-bowl probability are obtained from the result of the image processing unit, and are used directly as membership values via one-to-one membership function mapping. The membership functions for the linguistic variables (warm, cold, hot) of temperature values and the centroid de-fuzzification are illustrated in Fig. 9.

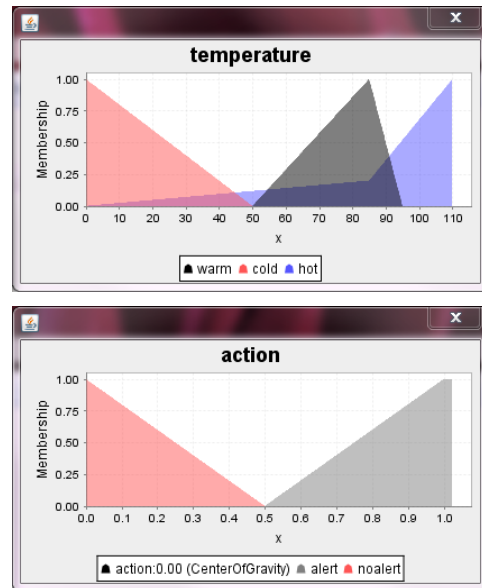


Fig. 9: The membership functions for evaluating temperature values and centroid de-fuzzification

Pet Activities Monitor will send an MSG 38 message if an anomalous condition exists. When an anomaly is detected, Alerter will send an alert to notify the pet owner, if no anomalous condition exists, the message MSG 38 will not be generated. Note that the fuzzy inference engine waits until it receives MSG 31 and MSG 34. To demonstrate the system, we

set the temperature to 80 degrees Fahrenheit and took an image of a Python Regius drinking from the water bowl to represent the captured image from Image Sensor, as shown in Fig. 10. The drinking probability is obtained from Image Processor, and indicates the drinking probability is 0.6 which is higher than the in-bowl probability. Therefore, the system detects the snake is drinking from the bowl.

When Alerter receives the message MSG 38 from Pet Activities Monitor to indicate that the condition is anomalous enough to warrant the user’s attention. When we check the e-mail inbox of the e-mail address specified in System Interface, we can find the email message describing that an anomaly is detected.

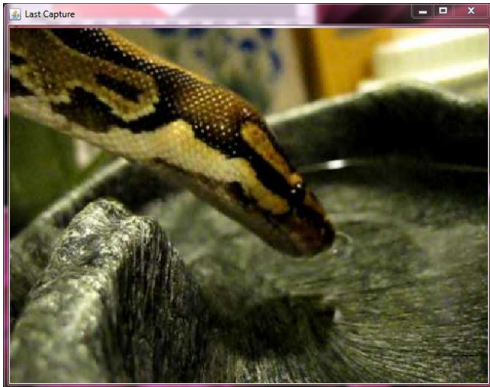


Fig. 10: The image used to represent the snake drinking from the water bowl.

4.2 Simulation for Pet Dog Care

Fig. 11 illustrates the information flow for the dog breed recognition. In this experiment, we use dog images as the pet dog in front of the webcam to demonstrate the effectiveness of the proposed system. Once the system recognizes the dog, the GUI will display the message contains the resulting dog breed less than 1/20 second. The system is working at 20 frames per second and can be adjusted according the hardware capability. The program is implemented as a joint effort of Java, OpenCV, Python, and C++ languages.

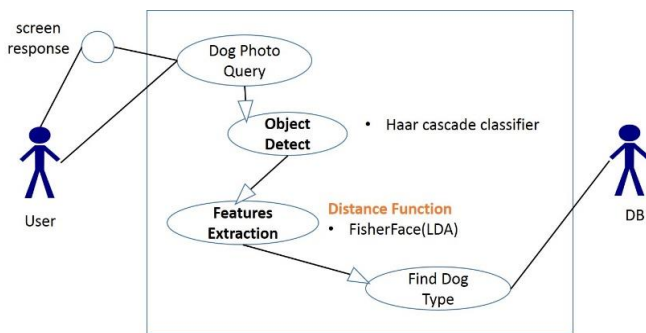


Fig. 11: The information flow for the dog breed recognition.

In this experiment, seven photos from four dog breeds, including the Eskimo, Shetland sheepdog, Borzoi, and Schnauzer are examined. A snapshot of the experimental results is shown in Fig. 12.

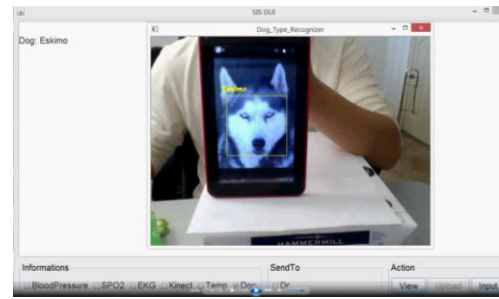


Fig. 12: A snapshot of the experimental results from a live demo.

V. CONCLUSIONS

In this study, a smart environment-aware pet care system has been established for providing pets with suitable living conditions by the fusion of heterogeneous sensors and the iterative slow intelligence computation. The proposed framework has component reuse and scalable features that make the proposed system can be easily extended and transferred to various pet care systems. The experimental test was performed using the SIS testbed and has shown that the proposed system could provide potential benefits for pet care.

While the system was currently developed in the simulation stage, it demonstrated the abilities to detect anomalous conditions and alert caretakers. The largest obstacle to generalizing the system is the specificity of the training image dataset required for the computer vision routines to work accurately. The results of this study provide a good case study in automatic caretaking which may have implications for autonomous healthcare systems for other animals. The scope of the system can be expanded by incorporating with spatially distributed sensors to monitor pet conditions and serve as a caregiver to manage the environmental conditions, watch for particular behaviors, feed pets and cooperatively pass messages through the network to specified locations.

REFERENCES

- [1] S. K. Chang, "A general framework for slow intelligence systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 20, pp. 1-15, 2010.
- [2] S. K. Chang, Yao Sun and Yingze Wang, "Component-based Slow Intelligence System", *Journal of Internet Technology* <http://jit.niu.edu.tw>, January 2013.
- [3] S. K. Chang, W. H. Chen, Bin Kao, L. Kuang, and Y. Z. Wang, "The design of pet care systems based upon slow intelligence principles," *Int'l Journal of Software Engineering and Knowledge Engineering*, 2014.
- [4] Paul Viola and Michael Jones, "Rapid object detection using a boosted cascade of simple features," In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp 511-518, 2001.
- [5] <http://vision.stanford.edu/aditya86/ImageNetDogs/>
- [6] P. N Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, v.19 n.7, p.711-720, July 1997.

A Slow Intelligence System Test Bed Enhanced with Super-Components

Shi-Kuo Chang, Sen-Hua Chang, Jun-Hui Chen, Xiao-Yu Ge, Nikolaos Katsipoulakis, Daniel Petrov and Anatoli Shein
Department of Computer Science
University of Pittsburgh, Pittsburgh, USA
{schang, sec104, juc52, xig34, nik37, dpp14, aus4}@pitt.edu

Abstract—The slow intelligence system (SIS) technology is a novel technology for the design of a complex information system that is aware of the environment through multiple sensors and capable of improving its performance over time. In this paper we describe a practical slow intelligence system test bed where super-components can be specified to describe interactions among components. These super-components are automatically transformed into time controllers for components so they can be managed by the SIS test bed. We illustrate our methodology on personal healthcare system design using this SIS test bed enhanced with super-components.

Keywords- slow intelligence system, environment-aware software engineering, super-component, SIS test bed, personal healthcare system.

I. Introduction

The slow intelligence system (SIS) is a general-purpose system characterized by being able to improve its performance over time in *iterative computation cycles* through a process involving *enumeration, propagation, adaptation, elimination* and *concentration*. An SIS continuously learns, searches for new solutions and propagates and shares its experience with peers [1].

The slow intelligence system (SIS) technology is a novel technology for complex information system design and a base for **Environment-Aware Software Engineering (EASE)**, which is the methodology and practice to design and/or improve a complex information system that is aware of the environment through sensors and capable of improving its performance over time in a changing environment. Such complex information systems have the following characteristics: *connected, multiple sourced, knowledge-based, personalized, hybrid* and *prodigious*.

The design of complex information systems faces the following challenges: (1) the operating environment, individual/collective user behavior and underlying technology base of such complex information systems are *constantly changing*; (2) there is *never a stable and static solution* for an “optimal” complex information system; and (3) there are *no general techniques* for the design of complex information systems. We believe that the SIS technology can be exploited to address these challenges.

There are many interesting theoretical issues concerning the design of slow intelligence systems [1]. To make the SIS technology useful to the practitioners, in this paper we describe a practical test bed for Slow Intelligence Systems enhanced with *super-components*, i.e., multiple components that can be activated either sequentially or in parallel and have complex interactions to search for better solutions. Furthermore these super-components can be automatically transformed into time controllers for components so they can be efficiently managed by the SIS test bed.

This paper is organized as follows. Section 2 introduces the essential characteristics of an SIS test bed enhanced with *super-components*. To illustrate our methodology, the essential components and super-components of a personal healthcare system are described in Sections 3 to 7. Background for slow intelligence system is presented in Section 8. Further research issues and applications of the SIS test bed to the design and analysis of *sentient networks* are discussed in Section 9.

II. SIS Test Bed with Super-Components

To design SIS-based systems, a practical SIS test bed is illustrated in Figure 2.1. The SIS test bed is a component-based software system. The center-piece of the test bed is the *SIS server* responsible for specification/creation/management of components and passing messages to/from components in the test bed. This test bed is implemented in Java and can run either under Windows or in the Eclipse environment.

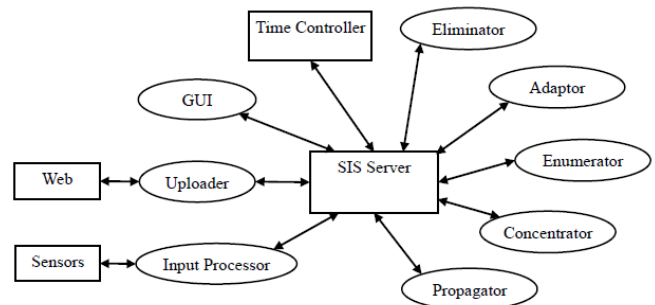


Figure 2.1. A slow intelligence system test bed.

The essential components of the basic SIS test bed include the *Graphical User Interface (GUI)* to interact with the end user, the *InputProcessor* to process input data from sensors and transform them into XML formatted messages, the *Uploader* to upload messages to the Internet, the *Propagator* to communicate with other SISs to propagate information and the *SIS operators suite (Enumerator, Adaptor, Eliminator and Concentrator)* to generate, adapt, eliminate and concentrate solutions. For the advanced SIS test bed enhanced with super-

components, there is also a *Time Controller* to initiate and control iterative computation cycles through *guard* predicates. The control and management of heterogeneous sensors requires a slow intelligence system with iterative computation cycles so that different sensors with different characteristics such as resolution, sampling rate, accuracy, etc. can be monitored and properly dealt with. During each computation cycle, the SIS operators suite is employed to optimize the processing of application data obtained from the environment through multiple sensors. The Time Controller determines the invocation and timing of the components in the computation cycles. A super-component is therefore a structured set of SIS operators to perform the computation cycles under the control of the Time Controller. A formal model of the computation cycle is introduced in [10]. To specify and create a super-component, a Create-Super-Component (CSC) message can be sent to the SIS server. An example of the CSC message structure is shown in Table 2.1.

MsgID:21	Description: Create Super Component	Example: Super component specification example
Variables:		
<ul style="list-style-type: none"> • Passcode (Passcode of Administrator) • SecurityLevel (Security Level of Administrator) • Name (Name of created Component) • SourceCode (Source code file name of created Component) • InputMsgID 1 (MsgID of first input message) • ... • InputMsgID k (MsgID of last input message) • OutputMsgID 1 (MsgID of first output message) • ... • OutputMsgID m (MsgID of last output message) • Component Description (Description of created Component in PNML or UML) • Component Var 1 (Variable name 1 of created Component) • ... • Component Var n (Variable name n of created Component) • KnowledgeBase (Name of knowledge base) 		

Table 2.1. The Create-Super-Component message structure.

In the above CSC message, the **component description** can be in the form of a PNML specification (if the computation model is a Petri net) or an XML document (if UML diagrams such as sequence diagrams, etc. are employed). In Table 2.2 a simple example of a (partial) PNML specification of a super-component is shown.

After a super-component is formally specified (such as Table 2.2), the Time Controller and other components of the super-component can be automatically generated from this specification. As mentioned above the super-component is controlled by the Time Controller, which sends messages to the constituent components to coordinate their execution. When a computation cycle is completed, the Time Controller decides whether to start another iteration of the computation cycle, or send messages to other super-components (or ordinary components) of the SIS system, depending on the guard predicate.

Multiple super-components with their respective Time Controllers may co-exist in an SIS system and interact with one another. The SIS server enhanced by the SC transformer is illustrated in Figure 2.2. All input messages except the new message 21 to create super-component are sent to the original SIS server. The new message 21 is processed by SC transformer that generates the Time Controller and sends a message 20 to SIS server to create the Time Controller component.

```

<?xml version="1.0" encoding="iso-8859-1"?>
<pnml>
<place id="P0">
<name>
<value>TimeController</value>
</name>
<initialCode>
<value>C:\\Users\\Steve\\Desktop\\SIS\\initialFile.java</value>
</initialCode>
<endCode>
<value>C:\\Users\\Steve\\Desktop\\SIS\\endFile.java</value>
</endCode>
</place>
<place id="P1">
<name>
<value>T1</value>
</name>
</place>
<place id="P2">
<name>
<value>T2</value>
</name>
</place>
<transition id="1001">
<name>
<value>T1toTimeController</value>
</name>
<orientation>
<value>1</value>
</orientation>
<code>
<value>C:\\Users\\Steve\\Desktop\\SIS\\othercode.java</value>
</code>
</transition>
<transition id="1002">
<name>
<value>T2ToTimeController</value>
</name>
<orientation>
<value>1</value>
</orientation>
<code>
<value>C:\\Users\\Steve\\Desktop\\SIS\\othercode.java</value>
</code>
</transition>
<transition id="1003">
<name>
<value>TimeControllerToT1T2</value>
</name>
<orientation>
<value>0</value>
</orientation>
</transition>
</pnml>

```

Table 2.2. A partial PNML specification.

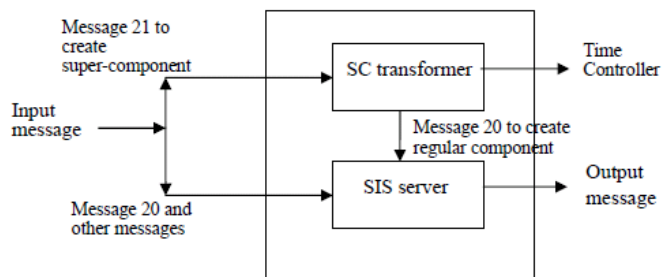


Figure 2.2. A super SIS server.

To illustrate the practical application of this methodology, an experimental personal healthcare system is shown in Figure 2.3. Although this is a specific application it nevertheless exhibits many important characteristics of a complex information system. Foremost among these characteristics is that heterogeneous multiple sensors are constantly changing due to technological advances or other reasons. Each monitor in a personal healthcare system can be a simple component in the simplest case, but more often than not it is a super-component to perform iterative computation cycles for the personal healthcare system. With our approach, the specification and upgrade of a super-components due to technological advances can be easily accomplished.

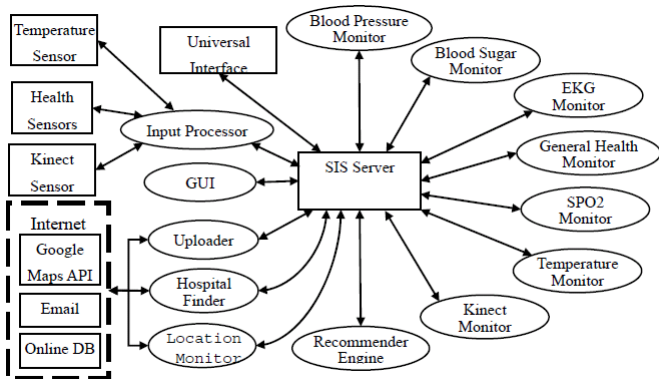


Figure 2.3. A personal healthcare system.

In the following sections we will describe the super-components, monitors and other novel components of the experimental personal healthcare system.

III. Temperature/Blood Pressure Super-Component

A personal healthcare system can assist a senior citizen who may not be computer-literate to deal with various monitors. For example, a *Temperature Monitor* can prevent a senior citizen from suffering from freezing or burning temperatures, and a *Blood Pressure Monitor* can monitor the person's blood pressure. With super-components, these monitors can exchange messages and work together to determine whether there is a need to send an alert message via the Internet to the Emergency Management System (EMS) or the responsible physician in case of an emergency. The situation is illustrated by Figure 3.1, which is a sub-network of Figure 2.3. The interacting monitors are in yellow color.

Once the SIS system is running, the GUI component is launched and the temperature settings such as start-monitor-time, end-time, refresh-time, high-temperature threshold and low-temperature-threshold can be set or adjusted, as shown in Figure 3.2.

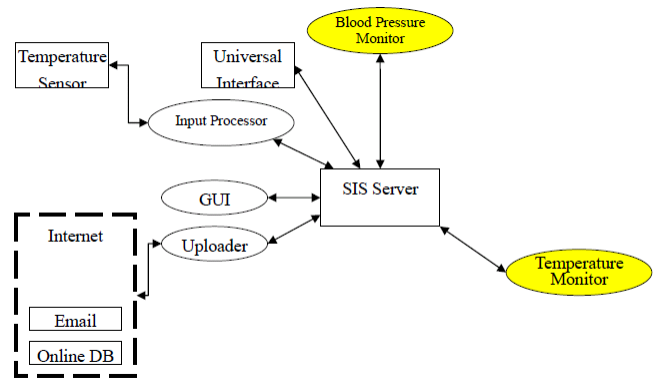


Figure 3.1. Interactions among Temperature and Blood Pressure Monitors.

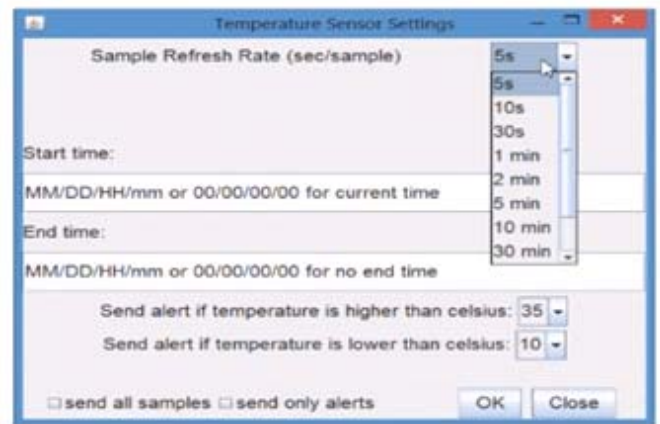


Figure 3.2. Temperature settings.

The Blood Pressure Monitor can then be launched to check whether the person's blood pressure is normal. In addition to working individually, these monitors can work together as a super-component to detect more complex conditions and upload and send *Complex Alert messages* to EMS as shown in Figure 3.3, where the e-mail contains the alert message that the blood pressure may not be normal perhaps due to the rising ambient temperature.

Complex Alert from BP_Component and Temperature_Component! Alert! Alert!

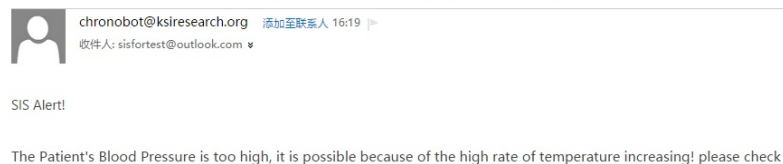


Figure 3.3. Complex alert from Blood Pressure Monitor and Temperature Monitor.

In Figure 3.4, the Petri-net description of a super-component involving the Temperature Monitor and the Blood Pressure Monitor is shown. The corresponding PNML specification can then be transformed into Time Controller to coordinate the interacting components.

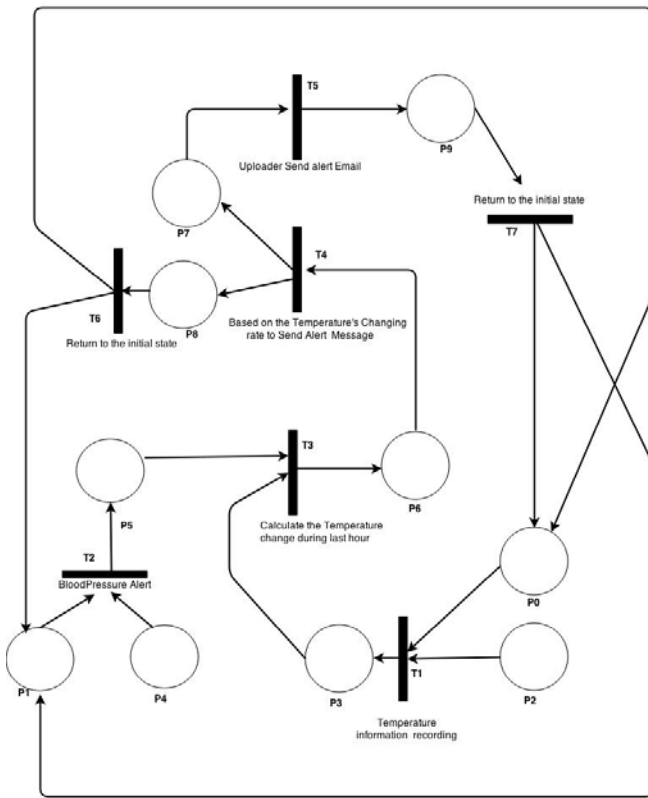


Figure 3.4. The Temperature/Blood Pressure super-component.

IV. Kinect/EKG Super-Component

A Kinect monitor is a component that accepts a series of messages from the Kinect sensor and sends out alerts to certain components when an emergency happens. The Kinect sensor and monitor together can detect and analyze motion patterns such as a person's fall (see Figure 4.1 and Figure 4.2), and the *fall detection* algorithm is incorporated into the Kinect monitor.

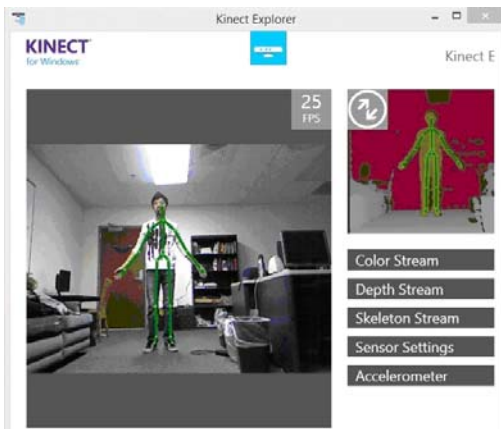


Figure 4.1. Skeleton figure of a person standing.

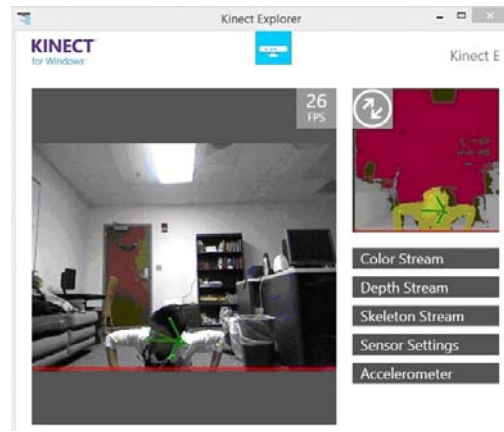


Figure 4.2. Skeleton figure of a person falling.

To achieve fall detection, we need to estimate the real-time position of the person. Kinect SDK software based upon Kinect sensor can track a person's skeleton consisting of more than 20 joints. Once we get a series of skeletons, we can easily extract the coordinates of joints. A frame is generated for each time interval, so we can calculate the difference between two consecutive frames and the speed of movement.

In the experimental personal healthcare system, we try to first track head movement and then estimate the positions of other joints to detect the motion of a person and in particular the fall of a person. When the position of the head cannot be reliably estimated, we can still use other available information to detect the person's movement pattern. We currently don't deal with multiple persons.

Fall detection alone is meaningless if it cannot be propagated and acknowledged by other components. We can send this information to the Uploader, which is the component responsible for collecting information from all other components and informing EMS and the physician in charge, and possibly building a knowledge base along the way. We can also send this information to other monitors responsible for the monitoring of different type of sensors so that they can make more accurate decisions. Likewise the Kinect monitor can also receive information from other monitors and work in a similar way.

As an example of such complex communication, if a person falls and the Kinect monitor also receives alerts from EKG monitor, it could mean this patient not only fell but also suffered heart problems. An alert can be generated either by Kinect monitor or EKG monitor or both, depending on the messaging sequence.

V. Location Monitor

In this section we discuss a Location Monitor for the SIS personal healthcare system, whose objective is to process the information about the location of the person, to track the

person's movements in real-time and to act accordingly in case of an emergency.

Dementia is a broad category of brain diseases. The number of patients, who suffer from dementia, is increasing in the United States of America for the last couple of years. Thus the mortal rate of fatalities, caused by dementia is increasing as well. The most common type of dementia is Alzheimer's disease. Some of the other more popular types are vascular dementia, Dementia with Lewy, Frontotemporal lobar degeneration, mixed dementia, Parkinson's disease and Creutzfeldt-Jacob disease. One in three seniors dies with Alzheimer's or another dementia. The statistics shows that 15.4 million caregivers provided an estimated of 17.5 billion hours of unpaid care, valued at more than 216 billion USD in 2012 [2].

The physicians measure what they call Clinical Dementia Rating (CDR), which changes between zero and three or more. The first stage is called CDR =0. There is no impairment for the patient at this stage. The next one is called Questionable Impairment; the value of CDR for it is -0.5. The third one is called Mild impairment and it is the last one, where the patient is capable of taking care of himself/herself. The value for it is 1. At this stage the patient gets geographically lost. For all CDR values beyond that point (2 and 3 – moderate and severe impairment), the person should have a personal caregiver. On the other hand, for all cases of CDR with value 1 or less, an automated monitor, such as the Location Monitor, can take over the responsibility of tracking the movements of the person and his/her location.

The Location Monitor communicates directly with four other components of the SIS Personal Healthcare System – GUI component, Input Processor component, Uploader and Hospital Finder component. The Location Monitor introduces four new messages to the system – GPS Reading, GUI Address Request, GPS Coordinates Request and GPS Coordinates Response. Location Monitor is using two different APIs of Google, Directions API and GmailAPI, to provide the desired functionality. The developers of Google Inc. provided a Java wrapper library for those APIs to be used.

Three basic scenarios involve the Location Monitor. The first one is the supply of data about the current location of the patient. The Input Processor receives the raw data from a sensor, which typically is a smart phone, which has a GPS receiver and some capability of reporting the data, obtained from the receiver, back to the system – Wi-Fi connection, GPRS, WCDMA, LTE or a combination of them. The Health Sensor sends the raw information, marshaled in a Sensor Data Input message. The Input Processor module processes it and sends the information further to the Location Monitor. The next scenario covers the case, when a physician sends the destination of the person to Location Monitor, which starts tracking the person. The third scenario involves the delivery of the last location of the patient, known by Location Monitor, to other components of the SIS system.

Once the Location Monitor receives an Address Request message, it contacts the Google Maps, using Directions API in order to receive a route for the person from his/her current location to the destination, received in the GUI Address Request message. The route contains a polyline, which is the smoothed polyline, which pins the route on the map. The Location Monitor tracks the location of the person for deviations from the received predefined route in the following way - upon every receiving of new location data information (i.e. GPS Reading message), the Monitor determines if the point, representing the received coordinates is within no more than 0.2 miles distance from every rectangle, formed by two consecutive points of the polyline of the route.

In case the location is outside the boundaries, the Location Monitor sends a short text message (SMS) to the person, informing him/her to stay where he/she is and letting him/her know that the help is on the way. An alert message is also sent to the uploader module and a mail is sent to the EMS and the physician on duty.

The Location Monitor was tested in SIS test bed for the Personal Healthcare system. The GPS Reading messages as well as the GUI Address Request messages were emulated with the Universal Interface component.

VI. Hospital Finder

Since there are an increasing number of sensors utilized to monitor the health conditions of senior citizens in today's world, it is possible to receive alert messages instantly when a person is in a dangerous state. This Hospital Finder component reacts to these messages by locating the person on the map, providing directions from the nearest hospitals to the person, and providing the contact information of these hospitals in order to rescue the person in a timely fashion. This component fits well with the rest of the Personal Healthcare System, and provides a useful enhancement that could potentially help save human lives. In what follows we will explain the system model of this Hospital Finder component and give an example of its operation using a real life scenario. The following scenario will utilize the Location Monitor component that monitors the person's location when he or she is out for a walk, and generates alert message 38 if the person's route drastically changes in an unexpected way.

When the Hospital Finder component receives alert message 38, it quickly reacts and opens the map based Graphical User Interface (GUI). The operator who is monitoring the person's health using the SIS system at this point knows that the person is in an emergency state. The operator should be looking for the closest hospitals to deliver help to the patient immediately.

The map-based GUI of the Hospital Finder component offers a variety of ways of locating the person. In this specific scenario, assuming the person has a GPS sensor with him/her, the easiest way to locate the person is by clicking the "Acquire GPS location" button. The other options include locating the

person (a) by address, (b) by coordinates, (c) by position on the map and (d) by GPS coordinates and so on.

In order to place the person's location on the map, the operator clicks the button "Acquire GPS location". The person's location is immediately requested with message 47 (Coordinates Request). The Location Monitor component receives this message and quickly sends the person's last location reading from the GPS sensor back to the Hospital Finder component. The incoming message 48 (Coordinates Response) is handled by the Hospital Finder component by populating the Latitude and Longitude fields on the map-based GUI, and placing an icon of the person on the map:

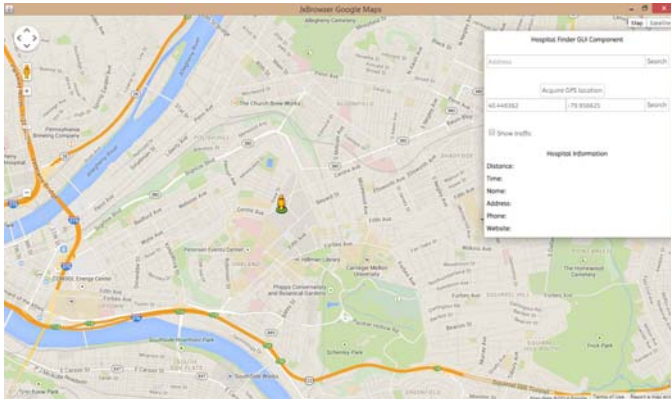


Figure 6.1. The person's last known location is displayed.

Now, since the operator has received the person's location, he or she can click the "Search" button next to the person's coordinates to find the closest hospitals:

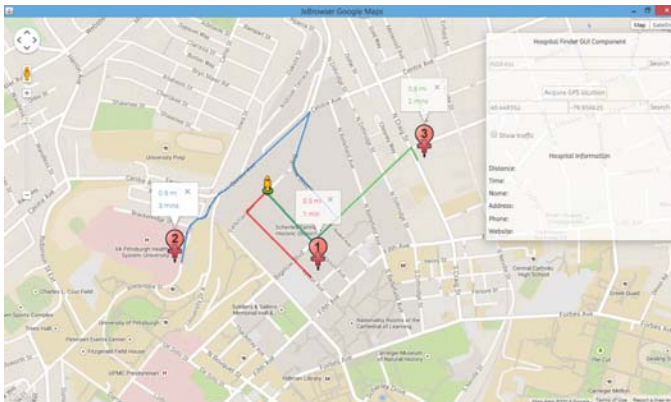


Figure 6.2. Closest hospitals with routes are displayed.

When the search button is clicked, the GUI requests the three nearest hospitals using the Google Maps Places API from the Google database. When the locations are received, the Hospital Finder component requests the directions from Google using the Google Maps Directions API to get to the person from these hospitals. As soon as the directions are received, the Hospital Finder component draws them on the map, and displays an information window above each one of the hospitals stating the distance and the time that it would take to get to the person from each hospital.

VII. Recommender Engine

The Recommender Engine for the Personal Healthcare System is capable of making elaborate decisions and proactively generate alert messages. This way, unwanted situations can be avoided in which the person may be in a state of imminent health deterioration. In what follows we will describe the design of the Recommender Engine, and usage scenarios in a real-life situation.

The Recommender engine waits for messages produced by sensors and mediated by the InputProcessor. Specifically, in its current state the Recommender Engine receives messages from: (a) blood pressure, (b) blood sugar, (c) EKG, and (d) SPO2 sensors. In the event that the Recommender Engine detects an imminent dangerous state, it produces alert messages and disseminates them in the SIS system.

The different components inside the Recommender Engine are as follows. First, the message parsing interface is responsible for handling input messages and produce alerts in the SIS network. The data transformation logic receives sensor data and turns them into a binary form that is readable by the recommender logic. The latter is responsible for building prediction models in order to implement the prediction logic needed for identifying dangerous situations in a proactive manner. Finally, the Recommender Engine keeps an internal storage module for storing user-defined Rules (conditions under which an alert should be generated) and pre-computed Prediction Models. Information are stored in the form of tuples (records) so that the system is able to predict dangerous situations.

The Recommender Engine works by using Collaborative-Filtering Algorithms to predict users' preferences. This approach is more generic compared to the Context-based approach of other recommendation systems. Hence, it can be easily modified to work with different scenarios. The only information to be stored should be tuples of the form:

user-id, item-id, preference

The user-id refers to a user showing interest (or a general connection) for a specific object (item-id). The interest can represent any kind of connection among two entities. For instance, it can represent how much a user likes a product, or it can represent that a user has had a characteristic represented by a specific id (object). Preference models the intensity of the connection of a user with an object. A preference can take values from 1 to 5, but it can also have a binary interpretation if a binary recommendation system is needed (i.e. yes or no answer). By forming the data accordingly, one can approximate any kind of situation and have the Recommender Engine produce successful predictions.

The Recommender Engine can be used when we need to predict dangerous situations for people. For any person we have sensor input for blood pressure, blood sugar, SPO2 and an EKG. The Engine's responsibility is to combine readings

from the aforementioned sensors, and by consulting user-defined rules, produce alert messages to the system. The Apache Mahout library (<http://mahout.apache.org>), which is a complete framework for recommendation systems, is used in implementation. Three scenarios are presented, each using different rules:

1. A patient is in alert if blood pressure and blood sugar are in near-dangerous levels.
2. A patient is in alert if blood pressure and SPO2 levels are in near-dangerous levels.
3. A patient is in alert if blood pressure, blood sugar and SPO2 levels are in neardangerous levels.

The Alert message produced by the Recommender Engine has the following form:

```
Name Value MsgID 64
Description Recommender System Alert
AlertType Recommender Alert
DateTime Current Date (i.e. "2014-10-30 15:05:10")
```

VIII. Related Work

The slow intelligence approach was first proposed by Shi-Kuo Chang [1]. In this section we will briefly review recently published papers in this area. The visual specification of component-based Slow Intelligence Systems is described in [3]. This work introduces the visual description of super-components by Petri nets or other UML diagrams. It provides the foundation of the present work. Component-based Slow Intelligence Systems has been applied to many areas, including social influence analysis [4, 5], topic and trend detection [6], high dimensional feature selection [7], image analysis [8], swimming activity recognition [9], and most recently pet care systems [10] and energy control systems [11]. In [10] the notion of an abstract machine for computation cycle was introduced. Our current approach is based upon it.

IX. Discussion

The super-component transformation algorithm can be extended to handle parallel/distributed processing of super-components in multiple computation cycles. At the implementation level we introduce one additional pair of tags, `<parallel>` and `</parallel>`, into the pnml specification to signify levels of parallel computation. Therefore the SC translator will append the super-component type as the suffix to the message that this transition represents. For example if the original message id is 1002 and a super-component `<parallel> 03</parallel>` is specified inside the related pair of transition tags of message 1002, then this message will become 1002.03. We can extend this technique to define `messagetype.SCsubtype.SCtype` and so on, so that messages are exchanged at different levels. At the theoretical level, we envision complex information systems as iterative slow intelligence systems with multiple and interacting computation cycles. In Wiener's Theory of General Resonance, he envisioned the interaction of multiple computation cycles. We

can call such general systems *Sentient Nets*. With the above proposed different levels of computation cycles and messages, we propose to continue the investigation of the properties of such general systems.

References

- [1] Shi-Kuo Chang, "A General Framework for Slow Intelligence Systems", International Journal of Software Engineering and Knowledge Engineering, Volume 20, Number 1, February 2010, 1-16.
- [2] Alzheimer's Disease Facts and Figures, www.alz.org/downloads/facts_figures_2013.pdf, 2013.
- [3] Shi-Kuo Chang, Yingze Wang and Yao Sun, "Visual Specification of Component-based Slow Intelligence Systems", Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, 2011, 1-8.
- [4] Shi-Kuo Chang, Yao Sun, Yingze Wang, Chia-Chun Shih and Ting-Chun Peng, "Design of Component-based Slow Intelligence Systems and Application to Social Influence Analysis", Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, 2011, 9-16.
- [5] Yingze Wang and Shi-Kuo Chang, "User Profile Visualization to facilitate MSLIM-model-based Social Influence Analysis based upon Slow Intelligence Approach", Proceedings of 2014 International Conference on Software Engineering and Knowledge Engineering (SEKE 2014), Vancouver, Canada, July 1-3, 2014.
- [6] Ji Eun Kim, Yang Hu, Shi-Kuo Chang, Chia-Chun Shih and Ting-Chun Peng, "Design and Modeling of Topic/Trend Detection System By Applying Slow Intelligence System Principles", Proc. of DMS2011 Conference, Florence, Italy, Aug. 18-20, 2011, 3-9.
- [7] Yingze Wang and Shi-Kuo Chang, "High Dimensional Feature Selection via a Slow Intelligence Approach", Proc. of DMS2011 Conference, Florence, Italy, Aug. 18-20, 2011, 10-15.
- [8] Shi-Kuo Chang, Li-Qun Kuang, Yao Sun and Yingze Wang, "Design and Implementation of Image Analysis System by Applying Component-based Slow Intelligence System.", Proc. of DMS2012 Conference, Miami, USA, Aug. 9-11, 2012.
- [9] Wen-Hui Chen and Shi-Kuo Chang, "Swimming Activity Recognition Based on Slow Intelligence Systems", Proc. of SEKE2013 Conference, Boston, USA, June 27-29, 2013.
- [10] S. K. Chang, W. H. Chen, Bin Kao, L. Kuang, and Y. Z. Wang, "The design of pet care systems based upon slow intelligence principles," *Int'l Journal of Software Engineering and Knowledge Engineering*, 2014.
- [11] Wen-Hui Chen and Shi-Kuo Chang, "Applications of Slow Intelligence Frameworks for Energy-Saving Control", Proceedings of 2014 International Conference on Software Engineering and Knowledge Engineering (SEKE 2014), Vancouver, Canada, July 1-3, 2014.

An Adaptive Contextual Recommender System: a Slow Intelligence Perspective

S.K. Chang, Duncan Yung
Computer Science Department
University of Pittsburgh
Pittsburgh, USA
{chang, duncanyung}@cs.pitt.edu

F. Colace, L. Greco, S. Lemma, M. Lombardi
DIEM
Università degli Studi di Salerno
Salerno, Italy
{fcolace, lgreco, slemma, mlombardi}@unisa.it

Abstract — This paper introduces an Adaptive Context Aware Recommender system based on the Slow Intelligence approach. The system is made available to the user as an adaptive mobile application, which allows a high degree of customization in recommending services and resources according to his/her current position and global profile. A case study applied to the town of Pittsburgh has been analyzed considering various users (with different profiles as visitors, students, professors) and an experimental campaign has been conducted obtaining interesting results.

Keywords — *Recommender System - Slow Intelligence Approach - Smart Adaptive System - Context-aware computing*

I. INTRODUCTION

Recommender Systems represent a meaningful response to the problem of information overload since the mid-1990s [28] when early works on this topic have been proposed. The aim of such systems is to predict user's preferences and make meaningful suggestions about items that could be of interest [29]. In literature, there are various approaches for recommending systems. In the content-based approach, the system recommends an item to a certain user relying on the ratings made by the user himself for similar items in the past [26]. In recent times, some improvements, such as a deeper user profile analysis [33] and the use of probabilistic methods [35], have been introduced together with some attempts to apply the content based approach to multimedia data [23, 18, 24]. However, a critical drawback of this approach is overspecialization, since the systems only recommend items similar to those already rated by the user. Another interesting approach is the collaborative filtering [1]. In this case, the recommendation is performed by filtering and evaluating items with respect to ratings from other users [33]. Ratings can be attributed in different ways and collected by explicitly asking users or implicitly tracking their actions [2]. Two basic methods, passive and active filtering, are exploited for filtering and recommending items together with nearest neighbor techniques [27, 21]. An important limitation of collaborative filtering systems is the cold start problem: situations in which a recommender is unable to make meaningful recommendations due to an initial lack of ratings.

A particular kind of collaborative approach is the collaborative competitive filtering that aims at learning user preferences by modeling the choice process in recommender systems [34]. Content-based filtering and collaborative filtering may be then combined in the so-called hybrid approach that helps to overcome limitations of each method [30]. In general, a recommendation strategy should be able to provide users with relevant information depending on the context [15, 19, 20] (i.e. user location, observed items, weather and environmental conditions, etc.) as in Context Aware Recommendation Systems. In the Contextual Pre-filtering techniques context information is used to initially select the set of relevant terms, while a classic recommender is used to predict ratings. In Contextual Post-filtering approaches context is used in the last step of the recommending process to contextualize the output of a traditional recommender. An important improvement for traditional recommender systems is in the possibility to embed social elements into a recommendation strategy [38]. In fact, the great increase of user-generated content in social networks, such as product reviews, tags, forum discussions and blogs, has been followed by a bunch of valuable user opinions, perspectives or tastes towards items or other users, that are useful to build enhanced user profiles. In such context, customer opinion summarization and sentiment analysis [39, 13] techniques represent effective improvement to traditional recommendation strategy, for example by not recommending items that receive many negative feedbacks [38]. Indeed, a lot of attention is nowadays being paid from vendors to consumer's voices because of the great influence they may have on the opinions and decisions of others [32] and some companies already provide several opinion mining services (e.g., Amazon, Epinions, etc.). In recent times, some works have been proposed to extend traditional collaborative filtering with the use of sentiment analysis techniques, thus providing effective improvement to system performances [22]: most of them make use of Part Of Speech (POS) tagging techniques and aim at refining standard collaborative filtering ranking outcomes in terms of numerical scales to take into account user community opinions. The work in [31] proposes a recommender system for movies that combines collaborative

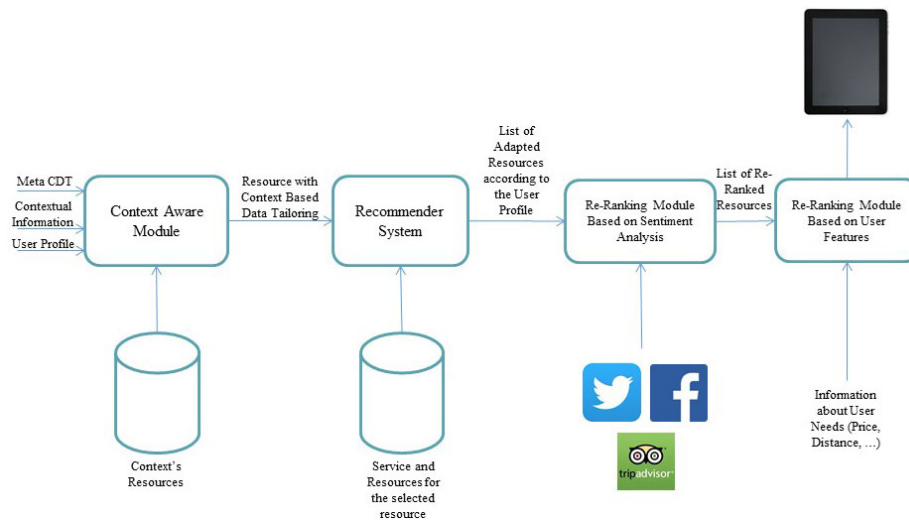


Figure 1 System architecture

filtering with sentiment: here sentiment classification is performed through both Naive Bayes classifier and unsupervised semantic orientation approach. Given this scenario, we propose a Recommender System based on the Slow Intelligence Approach [10]. The Slow Intelligence System is a general-purpose system characterized by being able to improve performance over time through a process involving enumeration, propagation, adaptation, elimination and concentration. A Slow Intelligence System continuously learns, searches for new solutions and propagates and shares its experience with other peers. Slow Intelligence Systems typically exhibit the following characteristics:

- Enumeration: In problem solving, different solutions are enumerated until the appropriate solution or solutions can be found.
- Propagation: The system is aware of its environment and constantly exchanges information with the environment. Through this constant information exchange, one SIS may propagate information to other (logically or physically adjacent) SISs.
- Adaptation: Solutions are enumerated and adapted to the environment. Sometimes adapted solutions are mutations that transcend enumerated solutions of the past.
- Elimination: Unsuitable solutions are eliminated, so that only suitable solutions are further considered.
- Concentration: Among the suitable solutions left, resources are further concentrated to only one (or at most a few) of the suitable solutions.

In the next paragraph, the general architecture of the recommender system is introduced and each module is described in details. In particular, the Context Aware Module and the Content Dimension Tree (CDT) approach is described. An example scenario and an experimental campaign close the paper.

II. A FRAMEWORK FOR CONTEXTUAL RECOMMENDATION BASED ON A SLOW INTELLIGENCE APPROACH

In this section, the System Architecture is introduced and described. Four modules compose the system: the context aware module, the recommender system module, Re-Ranking Module Based on Sentiment Analysis and the Re-Ranking Module Based on User Features. The architecture, depicted in figure 1, reflects the Slow Intelligence approach; in particular the context aware module and the recommender system module implement the adaptation and the enumeration phases, the Re-Ranking Module Based on Sentiment Analysis implements the propagation phase, the Re-Ranking Module Based on User Features implements the elimination phase. A “Contextual App” implements the concentration phase. In the following paragraphs more details about the various modules will be given.

A. The Enumeration and Adaptation stage

This stage aims at defining the user’s problems and the main strategies that have to be pursued for solving them. The combination of the Context Aware Module (CAM) and of the Recommender system can implement the Adaptation and the Enumeration phases. In the following paragraphs the two modules will be described:

1) The Context-aware module(CAM)

The purpose of this module is to provide a mechanism of dynamic and automatic invocation of services according to the context[11]. Since the purpose of this module is to deal with contextual changes that occur at runtime, there should be a mechanism that is concerned with the choice of the item to be invoked during the execution of the specific instance of the program, instead of associating a specific and concrete item to every activities in the design phase. Dynamic invocation of

items is implemented by context aware module configuration, in response to a user abstract request and according to the measured parameters at runtime. The concrete item to be invoked is chosen during the execution of the application. In general, we can divide the CAM module in two submodules: the High Level and the Low Level modules. The first one defines the problem and the resources or service at a high level of abstraction, while the second one gives the resources that contains concrete items or services.

The inputs of CAM are:

- Contextual Information: the user’s position, which is collected from GPS sensor.
- Resources: all the resources that can furnished in each context. The resources are identified by the definition of a Point of Interest.
- User Profile: user’s information, which are collected during the registration phase, e.g. user interests.
- Context Dimension Tree [3,4] is a model used to represent context in an extensible and orthogonal way, using the 5W-1H method. We realize a generic CDT, named Meta CDT, for all possible contexts and a specific CDT on the basis of the resource user choice.

For representing the scenario, the Context Dimension Tree (CDT) model shown in Fig.2 was used: the structure consists of two different types of nodes - the *black nodes*, which describe the context dimensions, i.e. the different points of view from which the system's situation can be observed, and the *white nodes* which describe the values that constitute the context; each node has a label with the name of the corresponding node. The CDT has a special white double circled node that represents the root of the tree. In addition, each leaf of the tree is a value node and they may feature parameters. The parameters are described by a white square and they are used as special filters helpful when a value has many instances.

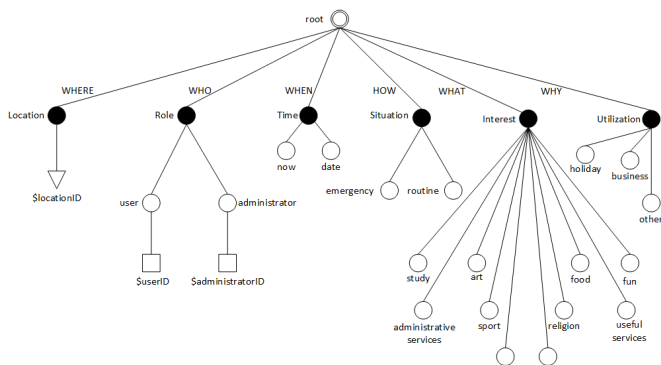


Figure 2 A Meta CDT example

This way of representing the context through the CDT allows the designer to characterize the relevant aspects of the considered scenario and to choose the dimensions and values of the tree in a correct way. It is important to underline how

the hierarchical structure of the CDT allows the description of the context with different levels of abstraction and granularity.

The output of CAM High Level is:

- Contextual Resources: all the resources that can furnished in a specific geographical area. User will choose a specific resource.

The input of CAM Low Level is:

- Resource: resource selected by user.

When the user chooses a resource, in CAM Low Level block, we know Location Dimension and we have a specific CDT with information associated to resource useful for representing current context and providing contextual services. The output of the system is a set of contextual items of current resource.

CAM module follows a methodology consists of three phases. Methodology has been realized in order to define all relevant contexts for the considered application, in order to provide contextual services managing database and performing reductions of their content based on the context.

- Design phase of contexts tree requires the design and use of CDT to represent and identify significant elements of context.
- Objective of definition phase of partial views is to identify each element of the context, then the value of each top dimension (child black node of root).
- Composition phase of views generates the global view associated with each context, which is made from union of partial views. The objective is to obtain a valid and specific query for the current context.
Then, it will be possible to interrogate the system in order to obtain the corresponding contextual items.

2) Recommender System

The developed recommender system is a content/service recommender system able to recommend a list of objects (contents or services), given the user profile and the contextual resource to which the objects belong. It can also dynamically update user profile. The resource is used for customizing our general recommender system to a particular, domain specific, recommender system. A resource can be a Cinema, a Museum, etc. The CAM sends to recommender system an identifier of the resource and through it the recommender system can recover, from a configuration database, some parameters that allow personalization of general recommender system in a domain specific way. In particular, every domain of interest has a specific database storing local informations and the similarity files, on objects of the domain, are generated once from a sub-system created ad hoc, as discussed later. CAM sends to recommender system also the user identifier. In fact, CAM maintains an high level user profile, with many interests for different domains, the recommender system instead maintains a low level user profile with specific

interests for the domain. These last interests are obviously a subset of the first. The proposed recommender system is actually an hybrid recommender system that has common features with Content-based and Collaborative systems. In particular it considers:

- User Similarity;
- Object Similarity;
- Users behaviour in the system;
- Users history.

We will now discuss each aspect in more detail.

User Similarity

User profile is used for clustering users and then creating groups of similar users. The clustering here is based on Jaccard Similarity [37] performed on user's interests for a specific domain. The interests for each user are represented through a binary vector. User similarity is a first important aspect to compute since it is fundamental for the calculation of the *global browsing matrix*, whose elements contain the ratio of the number of times object o_i has been accessed by any user immediately after o_j to the number of times any object in O has been accessed immediately after o_j .

Object Similarity

Object similarity is based on high-level information of the contents/services to recommend. A sub-system computes similarity indexes between objects through Wu-Palmer's metric[2]. For every domain there is a taxonomy, describing the most important features of the objects, that allows objects' comparison. The used metric allows to compute a similarity matrix (similarity files that are generated off-line). The object similarity is very important not only for the evaluation of the candidates but also for the location of the set of good candidates for recommendation.

The proposed recommender system is however independent from the way of computing the object similarity, so the sub-system designed, using taxonomies, is only one of the possible ways for computing object similarity.

Users behaviour in the system and Users history

The recommender system uses different structures to maintain users behaviour and users history. In particular, the following matrices are defined:

- local Connection matrix. Its generic element is defined as the number of times the object o_i has been selected by user u_i immediately after o_j .
- global Connection matrix. Its generic element is defined as the number of times object o_i has been selected by any similar user immediately after o_j .

User's history and behaviour are stored and retained in the local Connection matrix, where the occurrences and the order of what user prefers are stored. The global Connection matrix, instead, is used to retain dynamically the general behaviour of the similar users in the system. User profile is updated through the observation of the behaviour of the user in the system and by inferring the interests associated with chosen objects. Practically we analyze the local Connection matrix and when a generic element goes over a certain threshold, we analyze the related objects. From these objects, the related interests are extracted and user profile is updated. So we obtain implicit feedback from user's behaviour in the system. Now that we have a complete view of the system, we can summarize the process of the recommendation. The CAM module enumerates some recommendable objects to the recommender module starting from the user position. For each object, the most similar objects are obtained from the similarity matrix and the system adds all the objects o_i that have been selected by user in two steps after o_j , the 'init' objects. On this set a ranking vector is calculated, based on global browsing matrix and similarity matrix through Power Method's invocation. Finally from the ranked list of objects, obtained after this step, the first 50 objects are selected as recommendations for the current user.

B. Propagation stage

This module aims to involve the experience of other peers in the resolution of the problems. In particular, various policies can be defined (for example: the system can consider only peers that have profiles similar to the user, or with similar interests). For the selection of point of interests and resources, we consider only those having good reviews from users with similar profiles. This result can be obtained by the use of a module based on the sentiment analysis approach. In particular, this module can work on reviews that can be collected from Tripadvisor, Facebook or Twitter.

At the end of this stage a list of re-ranked resources/services is obtained.

1) Re-Ranking Module Based on Sentiment Analysis

This subsection describes the proposed methodology for the sentiments' extraction from user comments/reviews and its integration in the proposed recommendation strategy.

In particular, the used sentiment extraction technique is an improvement of the approach presented by some of the authors in a previous work [5], where the Latent Dirichlet Allocation (LDA) has been adopted for mining the sentiment inside documents. In our view, the knowledge within a set of documents can be represented in a compact fashion by the use of a complex structure: the Mixed Graph of Terms (mGT). This graph contains the most discriminative words and the

probabilistic links between them. More in details, we define a structure made of weighted word pairs, which has proven to be effective for sentiment classification problems as well as text categorization and query expansion problems [7, 8, 9]. The main reason of such discriminative power is that LDA-based topic modeling is essentially an effective conceptual clustering process and it helps discover semantically rich concepts describing the respective affective relationships. Using these semantically rich concepts, that contain more useful relationship indicators to identify the sentiment from messages, it is possible to accurately discover more latent relationships and make fewer errors in the predictions. The mGT is built starting from a set of comments belonging to a well-defined knowledge domain and manually labeled according to the sentiment expressed within them. In this way the mGT contains words (and their probabilistic relationships) which are representative of a certain sentiment for that knowledge domain. The LDA approach allows to obtain an effective graph by using only few documents. A mGT graph includes two kinds of nodes: the *aggregate roots* nodes, defined as the words whose occurrence is most implied by the occurrence of all other words in the training corpus, and the *aggregate* nodes, defined as the words most related to aggregate roots nodes from a probabilistic point of view.

In [5] the LDA approach and the mGT formalism have been used for the detection of sentiment in tweets. The approach aims at using the mGT, obtained by LDA based analysis of tweets, as a filter for the classification of the sentiment in a tweet. The sentiment extraction is obtained by a comparison between document and the *mixed graph of terms* according to the following algorithm:

• *Input of the algorithm:*

- A set of comments, reviews about items or social posts;
- The sentiment oriented mixed graphs of terms mGT+ and mGT- obtained analyzing the (positively and negatively) training comments;
- An annotated lexicon L.

• *Output of the algorithm:*

- The average probabilities P + and P - which express the probability that a sentiment, extracted from the set of comments or posts, is “positive” or “negative”.

• *Description of the main steps:*

1. For each word in the mGT+ and the mGT- their synonymous are retrieved through the annotated lexicon L.
2. For each comment c_i the probabilities $P_{c_i}^+$ and $P_{c_i}^-$ are determined as:

$$P_{c_i}^{+/-} = \frac{(A + B + C + D)}{4}$$

A being the ratio between the sum of occurrences in the comment of words that are Aggregate Root Nodes and the

total number of the Aggregate Root Nodes in the (positive/negative) mGT; B the ratio between the sum of occurrences in the document of words that are Aggregate Nodes and the total number of the Aggregate Nodes in the (positive/negative) mGT; C the ratio between the sum of the co-occurrence probabilities of Aggregate Root Nodes pairs that are in the document and the sum of all the co-occurrence probabilities of Aggregate Root Nodes pairs in the (positive/negative) mGT; D the ratio between the sum of the co-occurrence probabilities of Aggregate Nodes pairs that are in the document and the sum of all the co-occurrence probabilities of Aggregate Nodes pairs that are in the (positive/negative) mGT; 3. For each item the probabilities P + and P - are determined as:

$$P^+ = \sum_i \frac{P_{c_i}^+}{num_of_comments}$$

$$P^- = \sum_i \frac{P_{c_i}^-}{num_of_comments}$$

C. Elimination

This module aims to find the best resource/service according to some contextual features that characterize the user. In this phase, we propose to maintain a global resource/service quality table that keeps tracks of the quality of all resources/services in different aspects (e.g. service quality, price, transportation etc.) and a top-k function for each user so as to reflect his/her own preferences. Based on the quality table and the top-k function, each resource/service input of this module is ordered based on its top-k score. Furthermore, the user is asked to offer feedback after visiting the resources/services. Then, the top-k function is refined based on the feedback. More details will be presented below.

1) Global Resource Quality Table

The global resource/service quality table contains scores of all resources/services in all pre-defined aspects. It reflects the quality of resources/services. The value of each aspects is the median of all feedbacks from all users. Median Voter Theorem states that setting the value to be the medians of n feedbacks can satisfy most people in the population, where n is the population size. We believe that a value that can satisfy most people can truly reflect the quality of a resource. Hence, when the number of feedbacks increase along with time, the global resource/service quality table can gradually reflect the true quality of resources/services. For example, the global resource/service quality table below contains 3 resources and 4 aspects (score of each aspect ranges from [0-10]). The input order is not part of the table. It is given as the input of this module.

Resource	Service Quality (SQ)	Price (P)	Transportation (T)	Content (C)	Input Order (IO)
Carnegie Museum	9	10	1	5	2
Botanic Garden	9	5	9	2	1
Cathedral of Learning	2	3	8	7	3

Table 1: Global Resource/Service Quality Table

2) Top-k Function and Top-k Score

The top-k function is a personalized function which reflects the preference of a user. It takes into account all aspects in the global resource/service quality table and the input order. Hence, the number of coefficients of the top-k function is the number of aspects in the global resource quality table plus 1 and the sum of all coefficient equals to 1. Initially, all coefficients have the same value.

For example:

$$f = a \times SQ + b \times P + c \times T + d \times C + e \times IO_Score$$

where a, b, c, d, e are coefficients and $a + b + c + d + e = 1$.
For a new user, we assume that $a = b = c = d = e = 0.2$.

The top-k score of a resource/service is computed based on the top-k function, every aspect of the resource/service, and the input order of the resource. The input order score (IO_Score) is computed based on Definition D1.

Definition D1 [Input Order Score]: IO_Score is defined as:

$$IO_score = \begin{cases} 10 - (i - 1) & \text{for } |R| \leq 10 \\ 10 - \left\lfloor \frac{10(i-1)}{|R|} \right\rfloor & \text{for } |R| > 10 \end{cases}$$

where $|R|$ is the size of resource/service input and i is the input order (IO in Table 1).

All input resources/services are ordered base on their top-k scores (high to low) and top-k resources/services are returned to users.

3) Update of Top-k Function based on User Feedback

After the top-k resources/services are sent to the user, the user is asked to offer feedback to the system. The user can choose to offer feedback of any aspects of any resource/service to this module. Based on the feedback, this module computes the difference of each aspect between offered feedback and values in the global resource/service quality table and average the difference for each aspect. We propose to increase/decrease the top-k coefficient based on the average difference. We first recomputed the value of aspects that needed to be refined using this formula- $\frac{avg_diff}{10} \times coefficient$. Then, we normalize all coefficients so as to make their sum to be 1.

D. Concentration

This module aims to shows the outputs obtained from the other khmodules of the system. In particular, by this module the user can interact with the personalized services and resources. In this scenario, the concentration module has been built as a mobile contextual app that collects the outputs of the others modules and offers them by a friendly interface. The main feature of this app is that for each user it shows the ability to change services and contents depending on the context in which he/she is located. The app can also send information to the other modules that can update their information and refine the selected contents and services.

III. EXAMPLE SCENARIO

In this paragraph an example of how the proposed approach works is provided. Luca is a researcher and he is going to Pittsburgh for a two days business trip. During his spare time in Pittsburgh, he wants to visit some places. He has on his smartphone the contextual app that implements the Adaptive Contextual Recommender System approach. When he arrives at Pittsburgh, the app collects his current location and sends this information to the server. According to his profile, the Context Aware Module (CAM) retrieves a list of possible domains of interest from the context's database. In other words, the Context Aware Module selects a set of the possible domains that can be interesting for the user in a certain context. In this case, for example, if Luca has interests in nature and history the system will furnish resources that are linked to these domains. If Luca selects "Nature" the system will furnish all the resources that are related to this domain (e.g. Parks, Museums, Historical Buildings). At this point Luca can select Museum and the Recommender Module returns a list of resources or services related to this topic (e.g. Carnegie Museum of Natural History, the Heinz History Center, a booking web-site and so on). Luca can reorder the list of resources by the use of the sentiment analysis module. In this case the resources and the services selected by the recommender system can be re-ordered according to the sentiment retrieved in internet about them. In particular, the sentiment analysis module collects posts related to the selected resources from the famous social network "trip advisor" or from their official web sites. The SIS module receives the list of recommendations from the sentiment analysis module and a top-k function of Luca is used to re-rank the result of all recommendations from all domains. The top-k function reflects Luca's needs and preferences, and it is updated gradually based on Luca's feedbacks. The top-k score is computed based on Luca's current top-k function and a global resource score table, which is updated according to users' feedback of recommendations. Given the list of recommendations, Luca's top-k function $y = 0.8 \times worth_visit + 0.2 \times price$ reasonable, and the global resource score table, the SIS module computes the top-k

scores of all resources in all domains and return recommendations to Luca.

IV. EXPERIMENTAL RESULTS

For the experimental stage, 50 user profiles have been considered. Each user profile is defined as a vector of interests and can be dynamically updated according to user choices or feedbacks. The following set of possible interests has been considered: study, sport, courses, administrative services, transport, religion, food, useful services, fun. In our experimental campaign, we assumed profiles to contain at most three main interests. We identified about 126 geo-localized resources and services in Pittsburgh area, grouped in 29 points of interests such as: *Cathedral of Learning, Sennott Square, Restaurants area, Petersen Events Center, The Pitt Shop, Barco Law Library, Holland Hall, Carnegie Museums, Carnegie Mellon University, Phipps Conservatory* and so on. Depending on user profile and position, the CAM module and the Recommender System module provided a set of ranked results corresponding to recommended services or resources for each user. This ranking has been first refined by the sentiment analysis module and then the top-50 results for 50 users were given as input for the Elimination module. We assumed the global resource/service quality table to contain 4 features - *service quality, price, transportation, and content*, and they were all initialized to 7/10. Every features of the top-k function of all users was initialized to 0.2 (There are actually 5 features- *service quality, price, transportation, content, and input order*). Before we generated the top-5 result for all 50 users, we modeled the evolution process of the SIS module:

1. We defined a set of high quality services.
2. We randomly generated feedback from a user.
3. The randomly generated value was discount by 30% if the service was not in the high quality service set.
4. Then, the global resource/service quality table and top-k function of the user were updated based on the feedback.
5. We did this for all 50 users and repeat 2-4 for 30 times for all 50 users.

By doing that, we could model the randomness of user feedback while still penalizing poor quality services/resources. After that, top-5 results for 50 users were obtained by using results from recommender module and the current global service quality table and top-k functions in the Elimination module.

Relevance assessment was made by 150 students from the University of Salerno grouped in 50 sets (one for each profile): each student in a group assigned a binary relevance level to each of the top-5 retrieved results for the given profile; in this way, the relevance of each item was assessed through a majority vote rule. Once relevance levels have been assigned to the retrieved results, information retrieval performance measures were used to assess the quality of system's output. In particular, precision@5 (Fig.3), average precision and standard deviation on precision values for different profiles were calculated as shown in Table 2.

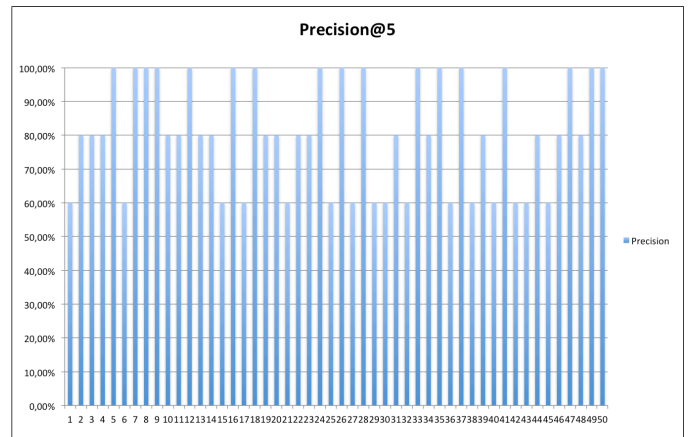


Figure 3 Precision@5 for each user profile

Average Precision@5	Standard Deviation
80,40%	16,41%

Table 2 Global performance evaluation

V. CONCLUSIONS

In this paper, an original approach to recommendation has been introduced. In particular, the proposed system is based on the Slow Intelligence Approach and integrates methodologies as the context aware approach and the sentiment analysis. The CDT formalism has been adopted for the context representation and a real case has been investigated developing a Contextual App for the Pittsburgh city. The results obtained by the experimental campaign are satisfying and show the good perspective of this kind of approach. Further developments involve the application of the proposed approach in various contexts and an improvement of the recommender approach according to an effective collaboration approach thanks to a closest integration with the most important social networks.

REFERENCES

- [1] Gediminas Adomavicius and Alexander Tuzhilin. *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. IEEE Transactions on Knowledge and Data Engineering, 17:734–749, 2005.
- [2] Massimiliano Albanese, Antonio d’Acierno, Vincenzo Moscato, Fabio Persia, and Antonio Picariello. *A multimedia recommender system*. ACM Trans. Internet Technol., 13(1):3:1–3:32, November 2013.
- [3] Bolchini, C., Schreiber, F. A., and Tanca, L. *A methodology for very small database design*. Information Systems, 32(1):61–82, March 2007.
- [4] Cristiana Bolchini, Carlo Curino, Fabio A. Schreiber, Letizia Tanca: *Context integration for mobile data tailoring*. SEBD 2006: 48-55.
- [5] Francesco Colace, Massimo De Santo, and Luca Greco. *A probabilistic approach to tweets’ sentiment classification*. In Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on, pages 37–42, 2013.

- [6] Francesco Colace, Massimo De Santo, and Luca Greco. *E-learning and personalized learning path: A proposal based on the adaptive educational hypermedia system*. International Journal of Emerging Technologies in Learning (iJET), 9(2):pp-9, 2014.
- [7] Francesco Colace, Massimo De Santo, and Luca Greco. *An adaptive product configurator based on slow intelligence approach*. Int. J. Metadata Semant. Ontologies, 9(2):128–137, April 2014.
- [8] Francesco Colace, Massimo De Santo, Luca Greco, and Paolo Napoletano. *Text classification using a few labeled examples*. Computers in Human Behavior, (0):–, 2014.
- [9] Francesco Colace, Massimo De Santo, Luca Greco, and Paolo Napoletano. *Weighted word pairs for query expansion*. Inf. Process. Manage., 51(1):179–193, 2015.
- [10] "Francesco Colace, Massimo De Santo"(2011). *A Network Management System Based on Ontology and Slow Intelligence System*. INTERNATIONAL JOURNAL OF SMART HOME. Vol. 5-3. Pag.25-38 ISSN:1975-4094.
- [11] Dey, A. K. *Understanding and Using Context*. Personal and Ubiquitous Computing 5, 1 (2001), 4-7.
- [12] Dey, A. K., and Abowd, G. D. *CybreMinder: A Context-Aware System for Supporting Reminders*. In Proc. HUC '00 (2000), pp. 172-186.
- [13] Xiaowen Ding, Bing Liu, and Philip S. Yu. *A holistic lexicon-based approach to opinion mining*. In Proceedings of the 2008 International Conference on Web Search and Data Mining, WSDM '08, pages 231–240, New York, NY, USA, 2008. ACM.
- [14] Ruihai Dong, Michael P. O'Mahony, Markus Schaal, Kevin McCarthy, and Barry Smyth. *Sentimental product recommendation*. In Proceedings of the 7th ACM Conference on Recommender Systems, RecSys '13, pages 411–414, New York, NY, USA, 2013. ACM.
- [15] Paul Dourish. *What we talk about when we talk about context*. Personal and ubiquitous computing, 8(1):19–30, 2004.
- [16] Andrea Esuli and Fabrizio Sebastiani. *Sentiwordnet: A publicly available lexical resource for opinion mining*. In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC 2006), pages 417–422, 2006.
- [17] Gayatree Ganu, Yogesh Kakodkar, and Am'elie Marian. *Improving the quality of predictions using textual information in online user reviews*. Inf. Syst., 38(1):1–15, March 2013.
- [18] Yoshinori Hijikata, Kazuhiro Iwahama, and Shogo Nishida. *Content-based music filtering system with editable user profile*. In Proceedings of the 2006 ACM symposium on Applied computing, SAC '06, pages 1050–1057, New York, NY, USA, 2006. ACM.
- [19] Katerina Kabassi. *Personalisation systems for cultural tourism*. In Multimedia services in intelligent environments, pages 101–111. Springer, 2013.
- [20] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. *Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering*. In Proceedings of the fourth ACM conference on Recommender systems, pages 79–86. ACM, 2010.
- [21] Yehuda Koren. *Factorization meets the neighborhood: a multifaceted collaborative filtering model*. In Proceedings of the 14th ACM SIGKDD International conference on Knowledge discovery and data mining, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.
- [22] Cane WK Leung, Stephen CF Chan, and Fu-lai Chung. *Integrating collaborative filtering and sentiment analysis: A rating inference approach*. In Proceedings of The ECAI 2006 Workshop on Recommender Systems, pages 62–66. Citeseer, 2006.
- [23] Veronica Maidel, Peretz Shoval, Bracha Shapira, and Meirav Taieb-Maimon. *Evaluation of an ontology-content based filtering method for a personalized newspaper*. In Proceedings of the 2008 ACM conference on Recommender systems, RecSys '08, pages 91–98, New York, NY, USA, 2008. ACM.
- [24] Katarzyna Musial, Krzysztof Juszczyszyn, and Przemyslaw Kazienko. *Ontology-based recommendation in multimedia sharing systems*. System Science, 34:97–106, 2008.
- [25] Nikolaos Pappas and Andrei Popescu-Belis. *Sentiment analysis of user comments for one-class collaborative filtering over ted talks*. In Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '13, pages 773–776, New York, NY, USA, 2013. ACM.
- [26] Michael Pazzani and Daniel Billsus. *Content-Based Recommendation Systems*. pages 325–341. 2007.
- [27] Naren Ramakrishnan, Benjamin J. Keller, Batul J. Mirza, Ananth Y. Grama, and George Karypis. *Privacy risks in recommender systems*. IEEE Internet Computing, 5:54–62, November 2001.
- [28] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. *GroupLens: An open architecture for collaborative filtering of netnews*. pages 175–186. ACM Press, 1994.
- [29] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.
- [30] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. *Methods and metrics for cold-start recommendations*. In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '02, pages 253–260, New York, NY, USA, 2002. ACM.
- [31] Vivek Kumar Singh, Mousumi Mukherjee, and Ghanshyam Kumar Mehta. *Combining collaborative filtering and sentiment classification for improved movie recommendations*. In Chattrakul Sombatheera, Arun Agarwal, Siba K. Udgata, and Kittichai Lavangnananda, editors, MI- WAI, volume 7080 of Lecture Notes in Computer Science, pages 38–50. Springer, 2011.
- [32] Johann Stan, Fabrice Muhlenbach, Christine Largeron, et al. *Recommender systems using social network analysis: Challenges and future trends*. Encyclopedia of Social Network Analysis and Mining, pages 1–22, 2014.
- [33] Xiaoyuan Su and Taghi Khoshgoftaar. *A survey of collaborative filtering techniques*. Advances in Artificial Intelligence, 2009, 2009.
- [34] Shuang-Hong Yang, Bo Long, Alexander J Smola, Hongyuan Zha, and Zhaohui Zheng. *Collaborative competitive filtering: learning recommender using context of user choice*. In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, pages 295–304. ACM, 2011.
- [35] Hilmi Yildirim and Mukkai S. Krishnamoorthy. *A random walk method for alleviating the sparsity problem in collaborative filtering*. In Proceedings of the 2008 ACM conference on Recommender systems, RecSys '08, pages 131–138, New York, NY, USA, 2008. ACM.
- [36] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. *Parallel matrix factorization for recommender systems*. Knowledge and Information Systems, pages 1–27, 2013.
- [37] Zheng, Nan, and Qiudan Li. *A recommender system based on tag and time information for social tagging systems*. Expert Systems with Applications 38.4 (2011): 4575-4587.
- [38] Xujuan Zhou, Yue Xu, Yuefeng Li, Audun Josang, and Clive Cox. *The state-of-the-art in personalized recommender systems for social networking*. Artif. Intell. Rev., 37(2):119–132, February 2012.
- [39] Li Zhuang, Feng Jing, and Xiao-Yan Zhu. *Movie review mining and summarization*. In Proceedings of the 15th ACM International Conference on Information and Knowledge Management, CIKM '06, pages 43–50, New York, NY, USA, 2006. ACM.

An Automated Testing Framework for Statistical Testing of GUI Applications

Lan Lin, Jia He, Yufeng Xue
Ball State University
Department of Computer Science
Muncie, IN 47396, USA
{llin4, jhe, yxue2}@bsu.edu

Abstract

It is known to be inherently more difficult and labor-intensive to functionally test software applications that employ a graphical user interface front-end, due to the vast GUI input space. We propose an automated testing framework for functional and statistical testing of GUI-driven applications, using a combination of two rigorous software specification and testing methods and integrating them with an automated testing tool suitable for testing GUI applications. With this framework we are able to achieve fully automated statistical testing and software certification. We report an elaborate case study that demonstrates a pathway towards lowered cost of testing and improved product quality for this type of applications.

1 Introduction

Software applications that employ a graphical user interface (GUI) front-end are ubiquitous nowadays, yet they present additional challenges to software testing. It is inherently more difficult and labor-intensive to functionally test a GUI-driven application than a traditional application with a command line interface, due to the vast GUI input space and the prohibitively large number of possible sequences of user input events (each event sequence being a potential test case) [13, 10, 11, 20]. Testing therefore needs to be automated in order to run a large sample of test cases to verify correct functionality.

In this paper we propose an automated testing framework for functional and statistical testing of GUI-driven applications, using two rigorous software specification and testing methods in combination, namely *sequence-based software specification* [12, 18, 19, 17] and *Markov chain usage-based statistical testing* [14, 16, 22, 21], and integrating them with an automated testing tool suitable for testing GUI applications, that provides fully automated statistical testing and software certification as a means to achieve high product

quality. Both methods and the supporting tools [1, 2, 3] were developed by the University of Tennessee Software Quality Research Laboratory (UTK SQRL). Although work has been done in the past to combine these methods together [8, 7, 9], it remains application and problem specific to work out a seamless integration from original requirements to fully automated statistical testing and software certification. We present in this paper our efforts and experiences along this path in solving a real world problem.

Sequence-based specification is a method for systematically deriving a system model from informal requirements through a *sequence enumeration* process [12, 18, 19, 17]. Under this process stimulus (input) sequences are considered in a breadth-first manner (length-lexicographically), with the expected system response to each input sequence given. Not all sequences of stimuli are considered since a sequence need not be extended if either it is illegal (it cannot be applied in practice) or it can be reduced to another sequence previously considered (the sequences take the system to the same state). Sequence enumeration leads to a model that can be used as the basis for both implementation and testing [8, 7, 9].

Markov chain usage-based statistical testing [14, 16, 22, 21] is statistical testing based on a *Markov chain usage model*. It is a comprehensive application of statistical science to the testing of software, with the population of all uses of the software (all use cases) modeled as a Markov chain. States of the Markov chain usage model represent states of system use. Arcs between states represent possible transitions between states of use. Each arc has an associated probability of making that particular transition based on a usage profile. The outgoing arcs from each state have probabilities that sum to one. The directed graph structure, together with the probability distributions over the exit arcs of each state, represents the expected use of the software in its intended operational environment. There are both informal and formal methods of building the usage model structure (sequence-based specification can be used as a formal method). The transition probabilities among states come

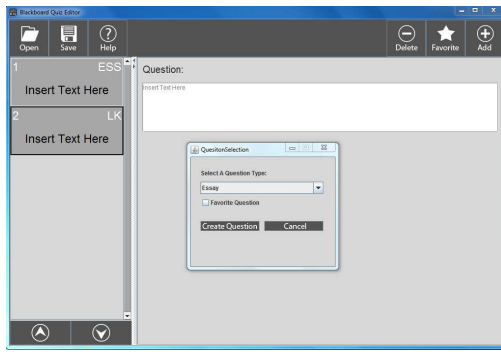


Figure 1. The BlackBoard Quiz Editor

from historical or projected usage data for the application.

The paper is structured as follows. The next section introduces our case study. Sections 3 and 4 illustrate how we constructed a usage model for statistical testing and present the model analysis results. Section 5 presents an automated testing framework we have developed for fully automated statistical testing of GUI applications, using the case study as a running example. Section 6 discusses our test and certification plan. Section 7 illustrates how testing was performed and presents results following a test case analysis. Section 8 concludes the paper.

2 The case study: The BlackBoard Quiz Editor (BBQE)

Our chosen case study is a Java GUI application, the BlackBoard Quiz Editor (BBQE, see Figure 1), that can author quizzes and save them in a format that can be imported in BlackBoard [4] - a learning management system used by Ball State University and many other institutions across the country for course delivery and management. The application was delivered in 2013 as a completed Ball State Computer Science major capstone project, considered of good quality by the client, and is being used by many faculty at Ball State. We were interested in automated statistical testing of this application to find bugs and to get a quantitative measure of its projected reliability.

The BBQE interface contains three main areas: a main toolbar, a quiz panel, and a question editor panel. It supports eleven question types, including the essay question type, the fill in the blank question type, the matching question type, the multiple choice question type, the true or false question type, the short answer question type, etc. Quizzes created in BBQE can be saved as a text file and easily imported into BBQE and BlackBoard.

To limit the testing problem to a manageable size, we defined our testing scope such that the System Under Test (SUT) contains only one question type: the essay question

Table 1. Stimuli for the BBQE under test

Stimulus	Long Name	Description	Interface	Trace
A	Add button	Click the add question button	Main window	req1
C	Create button	Click the create question button	Question creation window	req2
CO	Copy question	Copy the question	Mouse	req4
CQ	Cancel button	Click the cancel question button	Question creation window	req2
D	Down button	Click the question down button	Main window	req1
DQ	Delete button	Click the delete question button	Main window	req1
E	Essay question	Select the essay question type	Question creation window	req2
F	Favorite button	Click the favorite question button	Main window	req1
FS	Favorite checkbox	Click (Check/Uncheck) the favorite check box	Question creation window	req2
H	Help button	Click the help button	Main window	req1
HC	Help cancel button	Click the help cancel button	Help window	req3
P	Paste question	Paste the question	Mouse	req4
QF	Question fill	Fill in the question edit box	Question editor panel	req6
U	Up button	Click the question up button	Main window	req1

type, but includes all GUI features. Other models can be constructed similarly addressing other question types.

3 Usage modeling

In order to develop a usage model for statistical testing, we adopted a formal approach and first developed a rigorous specification (that encodes a formal system model) based on the requirements. As detailed requirements for the BBQE could not be found, we re-engineered requirements based on the user manual and the delivered application. We applied sequence-based specification [12, 18, 19, 17] and the supporting tool, the REAL [2], to developing a rigorous specification of the SUT.

The resulting specification contains 14 stimuli (inputs), 14 responses (outputs), 48 distinct states, 673 enumerated stimulus sequences, 9 original requirements, and 13 derived requirements. Figure 2 shows all the original (the top 9) and derived requirements for the SUT. Table 1 and Table 2 list all the stimuli and responses, respectively, across the system boundary. Each stimulus (input) and response (output) is given a short name (see the first columns) to facilitate sequence enumeration, tied to an interface in the system boundary, and traced to the requirements. Excerpts of an enumeration for the SUT are shown in Table 3. Stimulus sequences are enumerated in length-lexicographical order (based on the alphabetical order of stimuli as shown in Table 1) following the enumeration rules. Stimulus sequences are represented by concatenating stimuli to string prefixes with periods in the Sequence and the Equivalence columns. Each row shows for an enumerated stimulus sequence what should be the software's response, if the sequence could be reduced to a prior sequence (based on whether they take the system to the same state), and traces to the requirements that justify these decisions.

From the completed sequence enumeration we obtained a state machine for the SUT. We further added a source state together with an arc from the source to the state represented by the empty sequence (λ), and a sink state together with an

Requirements

req1: The GUI contains 8 buttons: Open to open a saved quiz, Save to save an edited quiz, Help to display the help window, Delete to delete an existing question, Favorite to add a question of the favorite question type to the quiz, Add to add a question to the quiz, Up to switch the current question with the question right above it, and Down to switch the current question with the question right below it.

req2: After the user clicks on the add button, the Question Selection window is displayed. The user can choose a question type from the dropdown list, check the check box that indicates whether the chosen question type is set as the favorite question type, click on the Create Question button to create a question of the chosen type, or the Cancel button to cancel creating the question. Double clicking the favorite question type checkbox is equivalent to unsetting it.

req3: After the user clicks on the help button, the help window is displayed. This button can be clicked anytime the application is running. The user can click the cancel button to close the help window.

req4: When the user right clicks on the question panel, there is a pop-up menu for more options (e.g., copying a question, pasting a question).

req5: If an operation can not be completed, there is a warning message.

req6: If a question gets created, the user can edit the question in the question edit box.

req7: No more than one question window exists at any time.

req8: No more than one help window exists at any time.

req9: If the question window is open, clicking all buttons in the main window except the help button has no response.

req10: If the user checks the favorite question type check box in the question window, and clicks the create button afterwards, the favorite question type will be set.

req11: If there is no question in the question panel, selecting the copy menu will have no response.

req12: If there is no question in the question panel, clicking the move down and move up buttons will have no response.

req13: If there is no question in the question panel, clicking the delete button will have no response.

req14: If the favorite question type has not been set, clicking the favorite question button will have no response.

req15: If no question has been copied, selecting the paste menu will have no response.

req16: The essay question type will be the default question type in the question window.

req17: Two stimulus sequences with the same sequence of events except clicking the help window (which could happen anywhere in the sequence) take the system to the same state.

req18: If there are more than one question in the question panel, after the user click the delete button, we get the response as there are question.

req19: If there is only one question in the question panel, deleting the question takes the system to the same state as when no question has been created.

req20: The user cannot click any button/checkbox/dropdown list if the window in which these items reside is not displayed. Likewise, if no question exists in the question panel, the user cannot edit the question edit box. A window cannot be closed without being opened first.

req21: Creating another question when there exists at least one question in the question panel does not change the system's state.

req22: The user issuing the copy operation multiple times has the same effect on system state as issuing the copy operation just once.

Figure 2. Requirements for the BBQE under test

Table 2. Responses for the BBQE under test

Response	Long Name	Description	Interface	Trace
EQC	Essay question created	Create an essay question	Question creation window	req2
EQFS	Essay question type set as favorite question type	Set the essay question type as the favorite question type	Question creation window	req2
EQS	Essay question favorite checkbox checked	Check the checkbox to set the essay question type as the favorite question type	Question creation window	req2
FSC	Favorite set checkbox unchecked	Uncheck the check box for favorite question set	Question creation window	req2
HW	Help window opened	Open the help window	Help window	req3
HWQ	Help window closed	Close the help window	Help window	req3
MD	Current question moved down	Move the current question down by one question	Main window	req1
MU	Current question moved up	Move the current question up by one question	Main window	req1
QC	Question copied	Copy the question	Right click menu	req4
QD	Question deleted	Delete the question	Right click menu	req4
QIP	Question input	Input the question	Question editor panel	req6
QP	Question pasted	Paste the question	Right click menu	req4
QW	Question window opened	Open the question window	Question creation window	req2
QWG	Question window closed	Close the question window	Question creation window	req2

Table 3. Excerpts of an enumeration for the BBQE under test

Sequence	Response	Equivalence	Trace
λ	0		Method
A	QW		req1, req2
C	ω		req20
CO	0	λ	req11
CQ	ω		req20
D	0	λ	req12
DQ	0	λ	req13
E	ω		req20
F	0	λ	req14
FS	ω		req20
H	HW		req3
HC	ω		req20
P	0	λ	req15
QF	ω		req20
U	0	λ	req12
A.A	0	A	req7
A.C	EQC		req2
A.CO	0	A	req9
...			
A.FS.C.CO.A.FS.H.A	0	A.FS.C.CO.A.FS.H	req7, req9
A.FS.C.CO.A.FS.H.C	EQFS, EQC	A.FS.C.CO.H	req2, req21
A.FS.C.CO.A.FS.H.CO	0	A.FS.C.CO.A.FS.H	req9
...			

arc from each state (except the source) leading to the sink. For the lack of compelling information to the contrary regarding the usage profile, we took the mathematically neutral position and assigned uniform probabilities to transitions in the usage model. The constructed usage model is diagrammed in Figure 3 using a graph editor (with 50 nodes and 619 arcs the visualization becomes very cluttered). Although not readable unless one zooms in, it illustrates the size of our testing problem.

4 Model analysis

We performed a model analysis using the JUMBL [3]. Table 4 shows the model statistics, including the number of nodes, arcs, and stimuli in the usage model, the expected test case length (the mean value, i.e., the average number of steps in a randomly generated test case) and variance.

The following statistics are computed for every node, every arc, and every stimulus of the usage model:

- **Occupancy.** The amount of time in the long run that one will spend testing a node/arc/stimulus.
- **Probability of Occurrence.** The probability of a node/arc/stimulus appearing in a random test case.
- **Mean Occurrence.** The average number of times a node/arc/stimulus will appear in a random test case.
- **Mean First Passage.** The number of random test cases one will need to run on average before testing a node/arc/stimulus for the first time.

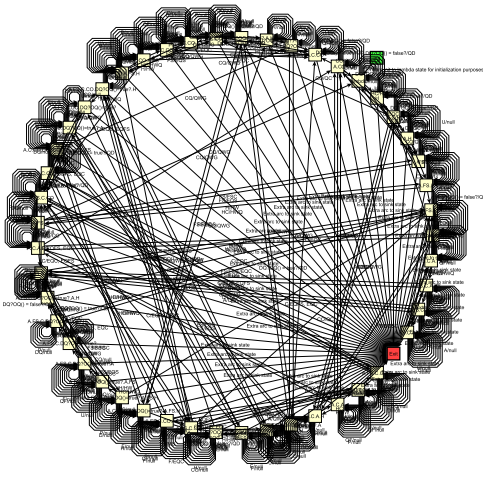


Figure 3. A state machine for the BBQE under test (with the source and the sink marked in green and red respectively)

Table 4. Model statistics

Node Count	50 nodes
Arc Count	619 arcs
Stimulus Count	18 stimuli
Expected Test Case Length	11.573 events
Test Case Length Variance	47.776 events
Transition Matrix Density (Nonzeros)	0.1056 (264 nonzeros)
Undirected Graph Cyclomatic Number	215

These statistics are validated against what is known or believed about the application domain and the environment of use.

5 An automated testing framework

Following the model analysis we developed an automated testing framework for fully automated, statistical testing of BBQE. This required (1) finding an automated testing tool suitable for our chosen application, and (2) integrating it with our statistical testing tool, the JUMBL, for automated test case generation, automated test case execution, and automated test case evaluation.

After some research we chose HP's Quick Test Professional (QTP) [5] as an automated testing tool for BBQE because it is HP's successor to its WinRunner and X-Runner software supporting functional and automated GUI testing and it also works for 32-bit machines. We created an object repository in QTP that registers all the static GUI objects of the SUT, and used QTP's Test Scripting Language (TSL) (a subset of VBScript) to write test cases that can run automatically in QTP.

Our constructed usage model was written in The

```

model bbq_rmm
[A]

"A/null" [A]
"c/EQC" [A.C]
"co/null" [A]
"co/nu11" [lambda]
"d/null" [A]
"do/nu11" [A]
"E/null" [A]
"Extra arc to sink state"[Exit]
"F/null" [A]
"FS/EQS" [A.FS]
"H/HW" [A.H]
"P/null" [A]
"u/null" [A]

[A.C]
"A/QW" [A.C.A]
"co/qc" [A.C.co]
...

```

Figure 4. An Excerpt of the usage model for the BBQE under test before annotation (written in TML)

Modeling Language (TML) [6] (Figure 4 shows a tiny piece). State names (representing usage states) are enclosed in square brackets and arc names (representing usage events/expected software's responses) are enclosed in quotation marks. For each state the list of event/expected response - next state pairs is given following the state name. For instance in Figure 4 there is an arc from state [A] triggered by usage event "A" with expected response "null" going back to state [A].

Using labels the TML model can include test automation information, which can be extracted when JUMBL automatically generates test cases of all types from the usage model using the GenTest command. The challenge was how we should annotate the states and arcs of the usage model with test scripts that could be understood by QTP such that when these test scripts are extracted and concatenated into a generated test case they literally become a program written in TSL that QTP could automatically execute. To achieve this we accomplished the following steps:

- **Stimulus generation with TSL.** We wrote TSL scripts that issue each possible stimulus (input) to the SUT. For instance, to issue the stimulus "A" (for clicking the add questions button in the main window), the following function is called:

```

Function stim_A()
  JavaWindow("Blackboard Quiz Editor") _
    .JavaButton("add").Click
End Function

```

- **Response checking with TSL.** We wrote TSL scripts

that check each possible response (output) (including the *null* response) is observed as expected. For instance, the following function is called to verify the response “EQC” (for having created an essay question in the question panel):

```
Function check_EQC()
  qnum = qnum + 1

  Set props = JavaWindow("Blackboard Quiz Editor") _
    .JavaStaticText("ESS(st)") _
    .GetTOProperties
  Set ChildObjects = JavaWindow( _
    "Blackboard Quiz Editor") _
    .JavaObject("JPanel_2") _
    .ChildObjects(props)
  ess = ChildObjects.Count

  If qnum <> ess Then
    returnValue = False
  Else
    returnValue = True
  End If

  qnum = ess

  check_EQC = returnValue
End Function
```

- **State verification with TSL.** We wrote TSL scripts that verify if the SUT is in any specific state as described by the usage model, probing values from the system’s state variables. For instance, the following function is called to identify if the system is in the state represented by the stimulus sequence λ (each such sequence is a canonical sequence in the sequence enumeration):

```
Function verify_lambda()
  If (check_var_EQ() = False) _
    And (check_var_EQS() = False) _
    And (check_var_HW() = 0) _
    And (check_var_QW() = 0) Then
    returnValue = True
  Else
    returnValue = False
  End If

  verify_lambda = returnValue
End Function
```

- **Usage model annotation.** We wrote Python code that automates the usage model annotation with TSL. Each state is annotated with a call to a state verification function. Each arc is annotated with a call to issue the stimulus to the SUT followed by one or more calls to check the observed responses. The rigorous specification serves as the test oracle. Each state/arc after annotation is associated with testing commands that are understood by QTP. When test cases are automatically generated and exported using the JUMBL, each test case literally becomes a TSL script that can automatically execute in QTP.
- **Result recording with TSL.** We wrote Python code that embeds TSL scripts in the annotated usage model recording test results. A test case is considered successful only if all its constituting steps are successful.

For any failed test case the test case number and all failure step numbers are written to a text file, together with information indicating whether each failure step is a continue failure (the rest of the steps can still run to completion after this failure step) or a stop failure (the following test steps included in this test case cannot be executed).

- **Automated test case execution and evaluation.** We wrote a shell script that runs from command line, automatically executes a large sample of generated test cases with our developed JUMBL-QTP interfaces, and records test results in a text file.
- **Reading failure data and recording it back in JUMBL.** We wrote a script that runs from command line, reads the failure data after testing is completed and records it back into the JUMBL for statistical analysis.

Figure 5 shows an excerpt of the usage model after annotation. Test automation information is included with a label (the text following a |\$ up to and including the end of line) in TML and an associated key (“a” in Figure 5 followed with a colon (:)) before the label). Test automation scripts can be attached to a model (e.g., the lines following declaring “model bbq_mml” up to the next empty line; here we do any needed test initialization and declare all stimulus generation/response checking/state verification functions), a state (e.g., the lines following declaring state “[A]” up to the next empty line; we verify if the SUT is in this state and if not record the last step/event as a stop failure and exit the test), or an arc (e.g., the lines following arc “A/null” up to declaring the to-state “[A]”; we issue the stimulus and check the response and if the observed and expected responses differ record the current step as a failure step).

Figures 6 and 7 show excerpts of an automatically generated test case from the usage model. Before model annotation the exported test case is a sequence of events/steps traversing the usage model starting from the source and ending with the sink (see Figure 6). After annotation all the test scripts associated with the states and arcs of the particular path are extracted and concatenated into a TSL script that is understood by QTP and automatically executable (see Figure 7).

6 A test and certification plan

We developed the following test and certification plan for the SUT:

- Run 48 *minimum coverage* test cases that cover every arc and every state of the usage model.

```

model bbq_mm1
a:| $If JavaWindow("Blackboard Quiz Editor").Exist(2)=true Then
| $ JavaWindow("Blackboard Quiz Editor").Close
| $End If
| $SystemUtil.Run "D:\BBQ.exe", "open"
| $
| $Function stim_A()
| $ JavaWindow("Blackboard Quiz Editor").JavaButton("add").Click
| $End Function
| $
| ...

[A]
a:| $If (verify_A) = False) Then
| $ currStamp = Cstr(testCaseNo) & "_" & Cstr(stepNo)
| $ If (StrComp(currStamp, stamp) <> 0) Then
| $ errorFile.write(stepNo)
| $ errorFile.write(" ")
| $ End If
| $ errorFile.write("s")
| $ ExitTest
| $End If

"A/null"
a:| $stepNo = stepNo + 1
| $record(0)
| $stim_A()
| $record(1)
| $If (check_null() = False) Then
| $ stamp = Cstr(testCaseNo) & "_" & "$s"
| $ errorFile.write($s)
| $ errorFile.write(" ")
| $End If

[A]

...

"Extra arc to sink state"
a:| $stepNo = stepNo + 1
| {$JavaWindow("Blackboard Quiz Editor").Close$}
| [Exit]

...

end // of model bbq_mm1

```

Figure 5. An Excerpt of the usage model for the BBQE under test after annotation

```

# =====
# Trajectory: 0
# Model: bbq_mm1
# Key:
# Method: random
#
# Events: 13
# Including failure information.
# =====
# Step: 1, Trajectory: 0
[init]."from init to lambda state for initialization purposes"
# Step: 2, Trajectory: 0
[lambda]."p/null"
# Step: 3, Trajectory: 0
[lambda]."U/null"
# Step: 4, Trajectory: 0
[lambda]."U/null"
# Step: 5, Trajectory: 0
[lambda]."U/null"
# Step: 6, Trajectory: 0
[lambda]."A/QW"
# Step: 7, Trajectory: 0
[A]."CQ/null"
# Step: 8, Trajectory: 0
[lambda]."D/null"
# Step: 9, Trajectory: 0
[lambda]."DQ/null"
# Step: 10, Trajectory: 0
[lambda]."D/null"
# Step: 11, Trajectory: 0
[lambda]."H/HW"
# Step: 12, Trajectory: 0
[H]."HC/HWQ"
# Step: 13, Trajectory: 0
[lambda]."Extra arc to sink state"

```

Figure 6. An example test case that is automatically generated from the usage model before model annotation

```

# =====
# Trajectory: 0
# Model: bbq_mm1
# Key:
# Method: random
#
# Events: 13
# Including failure information.
# =====
# Step: 1, Trajectory: 0
If JavaWindow("Blackboard Quiz Editor").Exist(2)=true Then
JavaWindow("Blackboard Quiz Editor").Close
End If
SystemUtil.Run "D:\BBQ.exe", "open"

Function stim_A()
JavaWindow("Blackboard Quiz Editor").JavaButton("add").Click
End Function

...

stepNo = 1
errorFile.write(vbNewLine & testCaseNo & " ")
stamp = Empty
currStamp = Cstr(testCaseNo) & "_" & "1"

# Step: 2, Trajectory: 0
If (verify_lambda) = False) Then
currStamp = Cstr(testCaseNo) & "_" & Cstr(stepNo)
If (StrComp(currStamp, stamp) <> 0) Then
errorFile.write(stepNo)
errorFile.write(" ")
End If
errorFile.write("s")
ExitTest
End If
stepNo = stepNo + 1
record(0)
stim_P()
record(1)
If (check_null() = False) Then
stamp = Cstr(testCaseNo) & "_" & "2"
errorFile.write(2)
errorFile.write(" ")
End If
...

# Step: 13, Trajectory: 0
If (verify_lambda) = False) Then
currStamp = Cstr(testCaseNo) & "_" & Cstr(stepNo)
If (StrComp(currStamp, stamp) <> 0) Then
errorFile.write(stepNo)
errorFile.write(" ")
End If
errorFile.write("s")
ExitTest
End If
stepNo = stepNo + 1
JavaWindow("Blackboard Quiz Editor").Close

```

Figure 7. An excerpt of an example test case that is automatically generated from the usage model after model annotation

- Run 200 *weighted* test cases that represent the 200 most probable paths of the usage model.
- Run 2,000 *random* test cases that are generated from the usage model based on the arc probabilities.
- Total testing consists of 25,577 transitions for the above 2,248 test cases.
- If all tests run successfully, this will demonstrate empirical evidence to support a claim of reliability > 0.90 given the defined protocol (of our usage model for the SUT, our selection of test cases, the actual result of testing, and the reliability model implemented in the JUMBL).

7 Automated statistical testing and test case analysis

Using the automated testing framework we had developed and the JUMBL, we were able to automatically generate, automatically execute, and automatically evaluate the sample of 2,248 test cases. Our testing was done on a laptop with Intel Core™ i-7-3630QM CPU with 4 cores, 2.40 GHz clock speed, and 8 GB memory. It took 2 days, 15 hours, 8 minutes and 4 seconds to run the 2,248 test cases, of which 710 were successful and 1,538 were failed.

We did a test case analysis using the JUMBL based on our testing experience. Excerpts of the test case analysis are shown in Table 5. Some important statistics include:

- **Nodes/Arcs/Stimuli Generated.** The number of states/arcs/stimuli covered in the generated test cases.
- **Nodes/Arcs/Stimuli Executed.** The number of states/arcs/stimuli covered in the executed test cases.
- **Arc/Stimulus Reliability.** The estimated probability of executing an arc / a stimulus in a test case successfully.
- **Single Event Reliability.** The estimated probability that a randomly selected arc can be executed successfully in a test case.
- **Single Use Reliability.** The estimated probability of executing a randomly selected test case successfully.
- **Optimum Reliability.** The estimated reliability if all generated test cases were executed successfully.
- **Relative Kullback Discriminant.** A measure of how close the performed testing matches the software use as described by the usage model.

Table 5. Excerpts of the test case analysis: Reliabilities

Single Event Reliability	0.726169681
Single Event Variance	2.72195572E-6
Single Event Optimum Reliability	0.985152717
Single Event Optimum Variance	394.383426E-9
Single Use Reliability	0.270545355
Single Use Variance	0.120506877
Single Use Optimum Reliability	0.907720385
Single Use Optimum Variance	38.7376831E-3
Arc Source Entropy	2.88 bits
Kullback Discrimination	0.6012524 bits
Relative Kullback Discrimination	20.879%
Optimum Kullback Discrimination	10.3936509E-3 bits
Optimum Relative Kullback Discrimination	0.360920256%

Of which the most important statistic, the *single use reliability*, estimates “the probability of the software executing a randomly selected use without a failure relative to a specification of correct behavior.” [15] The low single use reliability observed in this example (0.270545355) was due to the high number of failed test cases (1,538 out of 2,248).

Tracing through some failed test cases we identified 12 discrepancies between the specification and the code (see Figure 8). This record of specification-implementation discrepancies will be helpful in locating and fixing bugs in the released code.

1. If the user clicks the add button twice in a row, BBQE displays two question windows (the second add button press should have a null response).
2. When the question window is open, clicking the delete question button deletes a question (it should have a null response).
3. When the question window is open, clicking the down button moves the current question (if it is not the last one on the list) down by one (it should have a null response).
4. When the question window is open, clicking the up button moves the current question (if it is not the first one on the list) up by one (it should have a null response).
5. When the question window is open, clicking the favorite question button creates a new question of the favorite question type (it should have a null response).
6. When the favorite question button is clicked, a new question (of the favorite question type) always gets created no matter whether the user has previously set the favorite question type since the beginning of running the application (BBQE should not remember the favorite question type between different runs).
7. If the user clicks the help button twice in a row, BBQE displays two help windows (the second button press should have a null response).
8. When the question window is open, BBQE should not allow editing a question.
9. When the question window is open, there should be no response when the user clicks the paste button.
10. There should be no response when the user clicks the paste button unless the user has previously copied a question. But sometimes BBQE pastes a question whether or not any copying has been done since the program started.
11. BBQE crashes/freezes if the following sequence of events happens: one starts the program, then right clicks and from the menu chooses “paste” (nothing should happen as no question has been copied), then presses the add question button (the question window should be displayed), then clicks the create question button (a new question should be created, however, the program freezes).
12. When the question window is open, there should be no response when the user clicks the copy button.

Figure 8. BBQE specification-code discrepancies

8 Conclusion

In this paper we demonstrated an automated testing framework for fully automated statistical testing of GUI applications. We applied two rigorous software specification and testing methods and the supporting tools, and integrated them with an automated testing tool suitable for GUI applications, and reported an elaborate case study. As the readers might find out, working on any non-trivial real world problem requires considerable efforts be made to work out all the details needed for fully automated testing with no human intervention, however, by the end of the process we have the ability of running large numbers of tests, as well as an automated testing facility for low-cost, quick-turnaround testing and re-testing. All the artifacts we have produced in this process, including the usage model, test oracle, JUMBL-QTP interfaces, testing records, test plans, test scripts, test cases, product measures and evaluation criteria, all become reusable testing assets. Our experiences demonstrated a pathway towards lowered cost of testing and improved product quality for this type of applications.

Acknowledgements

This work was generously funded by Lockheed Martin Corporation and Northrop Grumman Corporation through the NSF Security and Software Engineering Research Center (S²ERC).

References

- [1] 2015. Prototype Sequence Enumeration (Proto_Seq). Software Quality Research Laboratory, The University of Tennessee. <http://sqr1.eecs.utk.edu>.
- [2] 2015. Requirements Elicitation and Analysis with Sequence-Based Specification (REALSBS). Software Quality Research Laboratory, The University of Tennessee. <http://sqr1.eecs.utk.edu>.
- [3] 2015. J Usage Model Builder Library (JUMBL). Software Quality Research Laboratory, The University of Tennessee. <http://sqr1.eecs.utk.edu>.
- [4] 2015. <http://www.blackboard.com>.
- [5] 2015. Quick Test Professional. Hewlett-Packard. <http://www8.hp.com/us/en/software-solutions/unified-functional-testing-automation/>.
- [6] 2015. The Modeling Language (TML). Software Quality Research Laboratory, The University of Tennessee. <http://http://sqr1.eecs.utk.edu/esp/tml.html>.
- [7] T. Bauer, T. Beletski, F. Boehr, R. Eschbach, D. Landmann, and J. Poore. From requirements to statistical testing of embedded systems. In *Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems*, pages 3–9, Minneapolis, MN, 2007.
- [8] L. Bouwmeester, G. H. Broadfoot, and P. J. Hopcroft. Compliance test framework. In *Proceedings of the Second Workshop on Model-Based Testing in Practice*, pages 97–106, Enschede, The Netherlands, 2009.
- [9] G. H. Broadfoot and P. J. Broadfoot. Academia and industry meet: Some experiences of formal methods in practice. In *Proceedings of the 10th Asia-Pacific Software Engineering Conference*, pages 49–59, Chiang Mai, Thailand, 2003.
- [10] T.-H. Chang, T. Yeh, and R. C. Miller. Gui testing using computer vision. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1535–1544, Atlanta, GA, 2010.
- [11] V. Chinnapongse, I. Lee, O. Sokolsky, S. Wang, and P. L. Jones. Model-based testing of gui-driven applications. In *Lecture Notes in Computer Science: Software Technologies for Embedded and Ubiquitous Systems*, volume 5860, pages 203–214, 2009.
- [12] L. Lin, S. J. Prowell, and J. H. Poore. An axiom system for sequence-based specification. *Theoretical Computer Science*, 411(2):360–376, 2010.
- [13] A. M. Memon and B. N. Ngyuen. Advances in automated model-based system testing of software applications with a gui front-end. *Advances in Computers*, 80:121–162, 2010.
- [14] J. H. Poore, L. Lin, R. Eschbach, and T. Bauer. Automated statistical testing for embedded systems. In J. Zander, I. Schieferdecker, and P. J. Mosterman, editors, *Model-Based Testing for Embedded Systems in the Series on Computational Analysis and Synthesis, and Design of Dynamic Systems*. CRC Press-Taylor & Francis, 2011.
- [15] J. H. Poore, H. D. Mills, and D. Mutchler. Planning and certifying software system reliability. *IEEE Software*, 10(1):88–99, 1993.
- [16] J. H. Poore and C. J. Trammell. Application of statistical science to testing and evaluating software intensive systems. In M. L. Cohen, D. L. Steffey, and J. E. Rolph, editors, *Statistics, Testing, and Defense Acquisition: Background Papers*. National Academies Press, 1999.
- [17] S. J. Prowell and J. H. Poore. Sequence-based software specification of deterministic systems. *Software: Practice and Experience*, 28(3):329–344, 1998.
- [18] S. J. Prowell and J. H. Poore. Foundations of sequence-based software specification. *IEEE Transactions on Software Engineering*, 29(5):417–429, 2003.
- [19] S. J. Prowell, C. J. Trammell, R. C. Linger, and J. H. Poore. *Cleanroom Software Engineering: Technology and Process*. Addison-Wesley, Reading, MA, 1999.
- [20] Z. U. Singhera, E. Horowitz, and A. A. Shah. A graphical user interface (gui) testing methodology. *International Journal of Information Technology and Web Engineering*, 3(2):1–18, 2008.
- [21] J. A. Whittaker and J. H. Poore. Markov analysis of software specifications. *ACM Transactions on Software Engineering and Methodology*, 2(1):93–106, 1993.
- [22] J. A. Whittaker and M. G. Thomason. A Markov chain model for statistical software testing. *IEEE Transactions on Software Engineering*, 30(10):812–824, 1994.

Test Model and Coverage Analysis for Location-based Mobile Services

Tao Zhang

School of software and
Microelectronic
Northwest Polytechnical University
Xi'an, China
tao_zhang@nwpu.edu.cn

Jerry Gao

Department of Computer
Engineering
San Jose State University
San Jose, USA
jerry.gao@sjsu.edu

Oum-EI-Kheir Aktouf

LCIS Laboratory
University of Grenoble,
France
Oum-EI-Kheir.Aktouf
@grenoble-inp.fr

Tadahiro Uehara

Software Innovation
Laboratories
Fujitsu Laboratories LTD.
Japan
uehara.tadahiro @jp.fujitsu.com

Abstract—Location-based services (LBS) are very important mobile app services, which provide diverse mobility services for mobile users anywhere and anytime. This brings new demands, issues, and challenges in mobile application testing. Today, mobile applications provide location-based service functions based on dynamic location contexts, mobile users and their travel patterns to deliver location-based mobile data, and service actions. Current software testing methods do not consider location-based validation coverage. Hence, there is a lack of research results addressing location-based mobile application testing. This paper focuses on mobile LBS testing. A novel test object model is proposed for quality validation of location-based mobile information services. In addition, the related test coverage metrics are also presented. These metrics can be useful for test engineers in designing test cases. A case study based on student testers is reported to demonstrate the potential application of the proposed model.

Keywords—component; Location-based service; mobile app; mobile testing; test model; test coverage; test metrics

I. INTRODUCTION

In recent years, more and more diverse mobile applications (mobile apps) have been developed to support different applications in social, news, tourism, health, business, and other domains. Hundreds of new apps are released daily, e.g. about 300 new apps appear on Apple's App Store each day. In turn, 750 million Android and iOS apps are downloaded each week [1].

Mobile Location-based services are services enhanced with positional data, which are provided by mobile apps using GPS, digit maps, and other techniques [2]. Many mobile apps provide interesting and convenient location-based services and functions. The mobile app Yelp (www.yelp.com) recommends nearby shops, restaurants, etc. In the social network mobile app Loopt (www.loopt.com), the users receive notifications whenever their friends are nearby. The mobile app Waze (www.waze.com) reports nearby traffic jams, policemen, gas stations and friends.

Mobile LBS have been identified as the one of most important features of mobile app services [3], and have become one hot research topic in mobile computing domains. In the recent years, there have been a number of published research papers addressing LBS. However, most papers primarily focus on the following areas:

- LBS infrastructure, framework, and architecture [4][5][6][7];
- Privacy preserving for LBS [8][9][10];
- Positioning techniques for LBS [11][12];
- LBS databases [13][14].

Till now, there are a few publications discussing mobile LBS testing. There is a lack of systematic and effective test models and methods for test engineers to address LBS testing. The test engineers often ask mainly the following questions about LBS testing:

- 1) *How to select and test a set of locations for LBS?*
- 2) *How to test different types of location objects?*
- 3) *How to test millions instances of location objects and their information?*
- 4) *How to test moving objects with dynamical moving paths and patterns?*

This paper focuses on cost-effective mobile LBS testing issues. The major contributions of this paper include four areas: 1) discussing the basic concepts, scope and challenges of mobile LBS testing; 2) proposing a novel test object model for mobile LBS testing; 3) some coverage metrics are defined for evaluating quality of mobile LBS testing; and 4) providing case studies to show the effectiveness of the proposed test model.

The rest of the paper is structured as follows. Section II discusses basic concepts and main challenges of LBS testing. Section III presents a novel test model and a test approach for location-based mobile information services. Section IV reports

the results of a case study. Section V reviews and discusses published papers and related research work. Finally, concluding remarks and future research directions are given in Section VI.

II. UNDERSTANDING MOBILE LBS TESTING

A. What is Mobile LBS Testing?

Location-based services are information, entertainment, services that are conveniently accessible by mobile users through GPS-enabled portable devices and mobile networks (e.g., 2G/3G/4G cellular telephones and Wi-Fi networks). Some organizations and scholars have defined LBS, and the classic definition is presented below.

Definition 1: LBSs are information services accessible with mobile devices through the mobile network and utilizing the ability to make use of the location of the mobile device [15].

Based on our recent literature survey, there is a lack of published papers on LBS testing for mobile apps, and also no precise definition of mobile LBS testing. Here we define it as below.

Definition 2: Mobile LBS testing refers to testing activities for native and Web apps on mobile devices to ensure quality in location-based functions, behaviors, information, and quality of service.

B. The Classifications of Mobile LBS

In recent years, mobile LBS have become more and more popular, and many mobile LBS apps are developed in various domains, such as navigation, information, emergency, advertising, tracking, games, management, and leisure. According to behavior characteristics of LBS, mobile LBS are divided into three types: *basic location services*, *location context information services*, and *location context interactive services*. Those services have different characters and features, and are compared in Table I.

TABLE I. MOBILE LBS SERVICES CHARACTERS

	Basic location services	Location context information Services	Location context interactive services
Objects	Mobile user	Location objects	Mobile objects
Positions	Moving along paths	Fixed positions	Moving along different paths
Typical Services	Map services, Location services, Navigation services	Search services, Access services, Update services	Interactive services
Service mode	Push way	Push way, Pull way	Push way, Pull way
Real-time feature	High	Low	High
Information	Digital map information	Location context information	Mobile Object properties
Domain	Map Navigation Emergency	Information Advertising Tracking Management Leisure	Game, Social network, Intelligent automobile

Basic location services - The mobile apps provide basic digital maps, location, and navigation services to help mobile users find correct ways to their destination. Some early map and navigation apps provide basic location services for mobile users.

Location context information services - The mobile users search nearby location objects and then access or update information of location objects. Those location objects have fixed positions, and provide static or dynamic information. Mobile users access information with push or pull ways. Most of information services are not real time services. The location context information services are widely used in many domains, for example, business, advertising, management, and leisure.

Location context interactive services - The multiple mobile objects interact with each other in same location contexts. Those mobile objects may be mobile users, smart cars, sensors, or other objects with positions. Location context interactive services provide different behaviors or functions according to locations or location relations between objects. Most of location context interactive services are real-time services, and are applied in some new domains, such as social networks, games and intelligent automobile.

Today, there are hundreds of mobile apps with location-based information services, such as, Yelp, Booking, etc. So this paper focuses on testing location-based mobile information services.

C. Why is Mobile LBS Testing Important?

Mobile LBS have been identified as killer services of mobile apps, which provide many conveniences and interests for mobile users. However, few published papers address the importance of mobile LBS testing. Here, its primary reasons and importance are listed as follows.

Reason #1: Multiple techniques for LBS - Mobile LBS normally use many different techniques, such as mobile computing, geographic information systems, GPS, cloud computing, Internet, etc. Those techniques make mobile LBS testing more complex.

Reason #2: Location impact for LBS - Location is the most critical factor for mobile LBS, which affects information, behaviors, functions, performance, dependability, privacy, etc.

Reason #3: Higher costs of LBS testing - Due to numerous locations, different kinds of location objects with changing information, and moving objects with unpredictable paths, the test engineers have to spend a lot of time and effort on mobile LBS testing.

D. Test activities and goals

Mobile LBS testing tends to focus on the following activities and goals:

LBS functionality and behavior testing - activities that validate location-based service functions, such as map and navigation services, information services, interactive services, etc.

LBS information and data testing - activities that validate LBS information and data, such as, positions, information, moving paths of mobile objects.

LBS QoS testing - activities that evaluate system load, performance, reliability, availability, scalability, and throughput in different location contexts.

LBS security and privacy testing - activities that check user data security and location information privacy.

LBS compatibility and connectivity testing - activities that assess mobile browser and platform compatibility, especially compatibility with different location techniques, and diverse wireless network connectivity.

LBS regression testing - Mobile LBS are updated frequently for fixing faults, adapting to new devices and platform versions, and enhancing functions. Regression testing is continuous for mobile LBS.

E. Scope of mobile LBS testing

LBS testing focuses on testing and validating location related functions and features of mobile apps. As shown in Figure 1, the scope of LBS testing includes the following six types of testing activities.

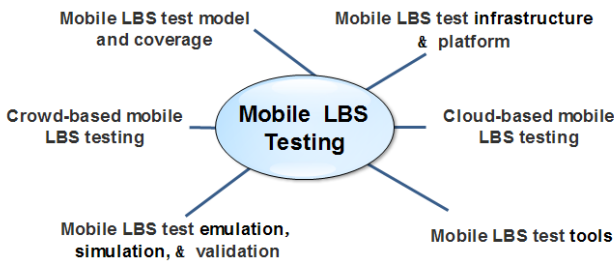


Figure 1. The Scope of Mobile LBS Testing

LBS test model and coverage - This refers to the activities that analyze and model main factors of mobile LBS, such as locations context, location object, location-related function and behavior, etc. The test models are used to generate test cases, and to analyze test coverage.

LBS test infrastructure and platform - This refers to the study of solutions on how to build the infrastructure and platform supporting automatic mobile LBS testing.

Crowd-based mobile LBS testing - This refers to the crowd-based test approach for mobile LBS, which allows to test mobile LBS in more extensive ranges.

Cloud-based mobile LBS testing - This refers to the cloud-based test approach for mobile LBS, which provides dynamic test resources and location simulation for mobile LBS testing.

LBS test emulation, simulation, and validation - This refers to study and apply different mobile LBS testing approaches.

Mobile LBS test tools - This refers to develop test tools to support automatic testing of mobile LBS.

III. THE TEST MODEL AND COVERAGE

This section presents a novel test model for location-based mobile information services. Based on this test model, a test approach and some coverage metrics are proposed.

A. The Mobile LBS Test Model

1) Basic concepts and definitions

First, some basic concepts about location contexts are defined. These concepts are important to design test cases for mobile LBS.

Definition 3: A location $I_i = (Long_i, lat_i)$ is a pair of real numbers representing the longitude $Long_i$ and the latitude lat_i of the location on the Earth surface.

Definition 4: The range $range_i(I_i, r_i)$ denotes one circular range, the center of the circular range is I_i , and the radius of the circular range is r_i .

Definition 5: A path $path_i(I_{i1}, I_{i2}, \dots, I_{ik})$ denotes a moving path through a set of location points (I_1, I_2, \dots, I_k) .

2) Test object model

The test object model for mobile LBS is proposed to represent the core objects and their relations. This model is used to design test cases and help test coverage analysis for mobile LBS. As shown in Figure 2, mobile LBS have four types of core objects: location context, Map, location object and location service.

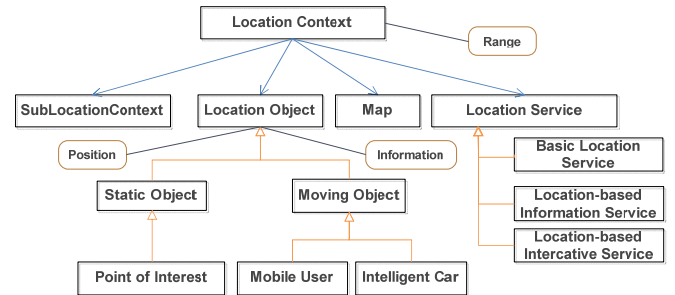


Figure 2. Test object model for mobile LBS

The location context LC for location-based mobile services is denoted as a 4-tuple

$$LC \left(SubLocationContexts, Map, LocationObjects, LocationServices \right).$$

Location context defines the range of mobile LBS, which includes map, a set of sub-location contexts, a set of location objects, and a set of location services.

Sub-location context is part of the location context, and represents a sub area. The location context contains many sub-

location contexts, and one sub-location context can contains some smaller sub-location contexts.

Map is used to display location contexts, and positions of location objects and mobile users. Map also has some basic behaviors, such as zooming, moving, *etc.*

Location Objects are a set of objects with positions. There are two types of location objects: static objects and mobile objects. Static objects have fixed positions, such as point of interest. Mobile objects have various moving patterns and paths. Location objects can provide static or dynamic information.

Location Services are a set of services in a location context. The main location context services include basic location services, location-based information services and location-based interactive services. Those location context services may have various behaviors in different location contexts.

B. The Test Coverage Metrics

In order to analyze and ensure test quality, some test coverage metrics are defined to evaluate mobile LBS testing.

Definition 6: The test range Rng_t is defined as the sum of all tested sub-location context's ranges. It is formulated as below.

$$Rng_t = \sum_{i=1}^n Rng_i$$

Here, Rng_i is the range of tested sub-location context SLC_i .

Definition 7: The range test coverage Cov_{reg} is defined as the ratio of the sum of all tested ranges Rng_t to all ranges Rng_a . It is formulated as below.

$$Cov_{reg} = \frac{Rng_t}{Rng_a} \quad (2)$$

In most situations, only a few ranges are tested because of restricted test resources. Then how to select and prioritize test locations and ranges is a critical issue for mobile LBS testing.

Definition 8: The location object type test coverage Cov_{LOt} is defined as the ratio of the number of tested location object types Num_{TLOt} to the number of all location object types Num_{ALOt} . It is formulated as below.

$$Cov_{LOt} = \frac{Num_{TLOt}}{Num_{ALOt}} \quad (3)$$

Location object type test coverage is used to ensure that all types of location objects are tested at least once.

Definition 9: The location object test coverage Cov_{LOi} is defined as the ratio of the number of tested location objects

Num_{TLOi} to the number of all location objects Num_{ALOi} . It is formulated as below.

$$Cov_{LOi} = \frac{Num_{TLOi}}{Num_{ALOi}} \quad (4)$$

Location object test coverage is used to test and evaluate information and behaviors of all location objects. However, there may be thousands of location objects in the location context, and then we have to select a small part of location objects to test.

Definition 10: The services test coverage Cov_{Srv} is defined as the ratio of the number of tested services Num_{TSrv} to the number of all services Num_{ASrv} . It is formulated as below.

$$Cov_{Srv} = \frac{Num_{TSrv}}{Num_{ASrv}} \quad (5)$$

Definition 11: The object services test coverage Cov_{OSrv} is defined as the ratio of the number of tested services for every type of location object Num_{TOSrv} to the number of all services for every type of location object Num_{AOSrv} . It is formulated as below.

$$Cov_{OSrv} = \frac{Num_{TOSrv}}{Num_{AOSrv}} \quad (6)$$

The same services may concern different types of location objects, and then we need test services for every type of location object individually.

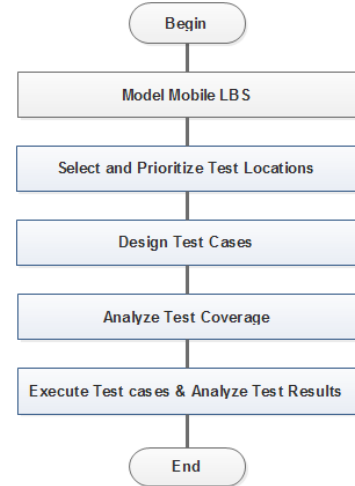


Figure 3. Model-driven test process for mobile LBS

C. Model-driven Test approach for Mobile LBS

The model-driven approach that we use for mobile LBS testing consists of five steps. The first step is to model location objects, location context Services for under-test mobile LBS

using the proposed meta-model. The second step is to select some test locations according to mobile user contexts. The third step is to design test cases according to the test models of mobile LBS. The fourth step is to analyze test coverage of mobile LBS. The last step is to execute test cases and analyze test results. This process is shown in Figure 3.

The main difficulty for mobile LBS testing is to select and prioritize test location contexts and location objects. Simulation-based and emulation-based test approaches are cost-effective to test different location contexts and location objects for mobile LBS. However, field test is necessary for mobile LBS. Then crowd-based testing is a novel and convenient approach to test a large number of location contexts and location objects for mobile LBS.

IV. CASE STUDY

To demonstrate the proposed approach for mobile LBS testing, we applied the approach to test one selected mobile app Tripadvisor. We conducted this case study in detail as described below.

A. The Test Model for Tripadvisor

Tripadvisor provides location-based travel services, which helps travelers to search and book hotels, flights, restaurants based on their locations. The test model for Tripadvisor is presented in Figure 4.

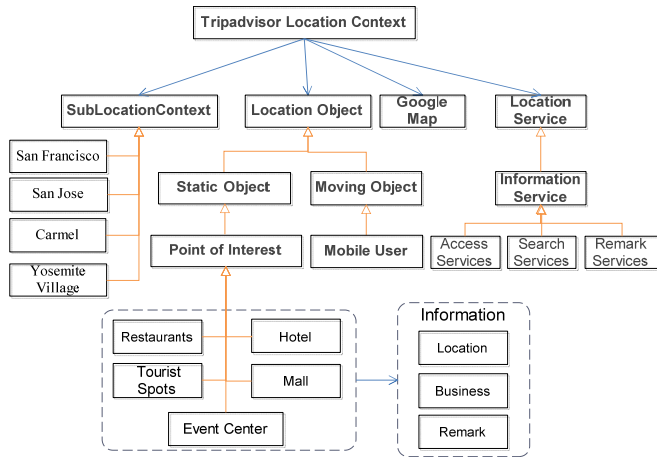


Figure 4. Test Object Model for Tripadvisor

TABLE II. TRIPADVISOR LBS DESCRIPTIONS

LBS		Description
Location context Services	Search Services	Search nearby location objects, such as: hotels, restaurants, tourist spots, malls, and event centers.
	Access Services	Access information of nearby location objects, such as positions, phone numbers, business hours, and remarks.
	Remark Services	Mobile users remark some nearby location objects by their experiences.

Tripadvisor provides some location objects, including restaurants, hotels, tourist spots, malls, and event centers. Those location objects have positions, remarks, and business information. Tripadvisor provides some services for mobile

users to search, access, and remark location objects. Those services are described in Table II.

B. Test approach for Tripadvisor

1) Selecting and specifying test locations

For testing LBS, we must select and specify test positions and ranges firstly. Tripadvisor has been used in many countries and cities, we only test Tripadvisor in San Francisco Bay Area because of restricted test resources. Four different locations are selected, which represent different kinds of location contexts. The details about selected location contexts are described in Table III.

TABLE III. LOCATION CONTEXTS FOR TRIPADVISOR

LC id	LC1	LC2	LC3	LC4
Location	San Francisco City	San Jose City	Carmel	Yosemite Village
Description	Big city	Medium city	Small city	Tourism area
Position	37.805623 -122.406722	37.808517, -122.411934	36.554986 -121.922041	37.742004, -119.582939
Range	600.6 km ²	466.1 km ²	2.798 km ²	3081 km ²
Restaurants	5097	1953	150	28
Hotels	233	73	23	30
Tourist Spots	290	62	18	57
Events	419	46	36	14
Malls	247	20	65	1
Sum	6289	2154	292	130

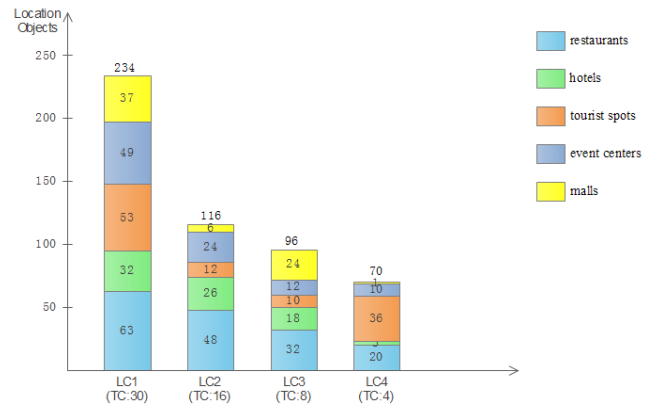


Figure 5. Tripadvisor test cases and tested location objects

2) Designing test cases for Tripadvisor

Based on location contexts, test cases (TC) are designed for mobile LBS testing. All types of location objects and location context services should be tested at least once. We designed 58 test cases for Tripadvisor LBS in 4 location contexts based on the proposed test model. The test cases and tested location objects are shown in Figure 5.

As shown in Figure 6, we design one test case for Tripadvisor to test searching nearby hotels in San Jose. Then the 29 hotel objects are shown in the map.

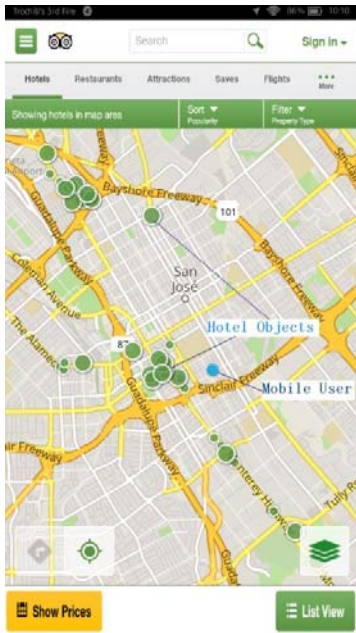


Figure 6. A test case example

3) Analyzing test results and coverages

The five types of location objects have been tested, and three location context services have also been tested. We only selected and tested four location contexts, so this is too small to calculate range test coverage. There are millions of location objects in Tripadvisor, we tested 516 location objects from 8862 located objects in the four selected location contexts. Finally, 4 faults have been found. The test coverages are described in Table IV, and a fault about losing mobile user position is shown in Figure 7.



Figure 7. A fault for Tripadvisor

TABLE IV. TRIPADVISOR LBS TEST COVERAGE

Test Cases	Cov_{LOt}	Cov_{LOi}	Cov_{Srv}	Cov_{OSrv}	Faults
58	5/5	516/8862	3/3	15/15	4

Using our proposed approach, the test engineers design test cases for testing more location objects and services based on the test model. According to table IV, all location objects and services are tested fully. This approach helps to reach high test coverage, to improve test effectiveness and save test costs.

V. RELATED WORK

Nowadays, mobile LBS have become one hot research topic in mobile computing domain, and many papers have been published to address different areas in mobile LBS.

Some mobile LBS frameworks and architectures have been proposed. Chengcheng Dai proposes a LBS framework using vehicular ad-hoc networks [4]. R. Gobi proposes a communication framework for data management in LBS [5]. A conceptual framework for personalized location-based tourism apps leveraging semantic web to enhance tourism experience is proposed by Mahmood [6]. Rui Jose proposes the AROUND architecture for supporting location-based services in the Internet environment [7].

The privacy issues are serious for LBS. Ben Niu proposes two dummy-based solutions to achieve k-anonymity for privacy-area aware users in LBSs by considering that side information may be exploited by adversaries [8]. K. G. Shin presents a comprehensive overview of the existing schemes for protecting LBS users' privacy [9]. An adaptive location privacy-preserving the system is presented, which allows a user to control the level of privacy disclosure with different quality of location-based services [10].

Position technique is the key for LBS. HuangChi Chen proposes a novel indoor positioning technique based on neural networks [11]. Al Nabhan presents a new strategy in achieving highly reliable and accurate position solutions fulfilling the requirements of Location-Based Services (LBS) pedestrians' applications [12].

Suprio Ray presents an in-memory database technique for location-based service, and introduces a parallel spatio-temporal index to support historical, past and predictive (future) location-based queries [13][14].

However, there are a few publications about mobile LBS testing. Jerry Gao discusses the issues and difficulties of mobile LBS testing [16]. Ke Zhai proposes a suite of metrics for prioritizing test cases for regression testing of LBS [17]. Huichun Chu proposes a two-tier test approach for location-aware mobile learning systems [18]. Solveig Bjørnstad presents an example study about evaluation of a location-based mobile news reader [19]. Jiang Yu analyzes test requirements, and presents a scalable testing framework for mobile LBS [20]. However, those papers do not discuss test models for mobile LBS.

We have proposed an initial test model for function services of mobile LBS [21]. In this paper, we improve and perfect the test model by considering location context, range, location objects, moving pattern and path, and some new metrics are defined for evaluating mobile LBS test coverage.

VI. CONCLUSION AND FUTURE WORK

Recently, mobile LBS have become popular among research groups. Because mobile LBS provide context-sensitive functions based on location information, this brings many new difficulties and challenges for mobile LBS testing.

This paper analyzes the main factors about mobile LBS testing. A new test object model has been proposed, and some metrics are defined to evaluate test coverage for mobile LBS. The test object model and test metrics help test engineers to design test cases for mobile LBS, and to improve test quality by higher test coverage.

Future research directions include four areas: a) selecting and prioritizing location contexts and location objects for mobile LBS testing; 2) enhancing the test approach to validate and verify correctness and timeliness of location-based services; 3) designing and specifying moving paths for mobile objects; and 4) developing automatic test tool supporting mobile LBS testing with emulation and simulation.

ACKNOWLEDGEMENT

This research project was supported by National Natural Science Foundation of China (Program No. 61103003). This research project was jointly funded by Fujitsu Research Lab. on Mobile SaaS Testing from 2013-2015.

REFERENCES

- [1] Adrian Holzera, and Jan Ondrusb. "Mobile application market: A developer's perspective", *Telematics and Informatics*, 2014, 28(1): pp.22-31.
- [2] Sergio Ilarri, Arantza Illarramendi, Eduardo Mena, Amit Sheth, "Semantics in Location-Based Services," *IEEE Internet Computing*, vol. 15(6), 2011, pp.10-14.
- [3] Shang-Pin Ma, Wen-Tin Lee, and Chia-Hsu Kuo, "Location Explorer with information services: A mobile application to deliver location-based web services", 2013 IEEE International Symposium on Next-Generation Electronics (ISNE 2013), pp.283 - 286
- [4] Chengcheng Dai, Chiyin Chow, and Jiadong Zhang. "Utilizing road-side infrastructure for location-based services in vehicular ad-hoc networks", the 8th International ICST Conference on Communications and Networking, 2013, pp.546-551.
- [5] R. Gobi, Dr. E. Kirubakaran, and Dr. E. George Dharma Prakash Raj. "ComFrame: A Communication Framework for Data Management in Mobile Location Based Services", *International Journal of Computer Science and Telecommunications*, vol.3(7), 2012, pp.65-68.
- [6] Mahmood, F.M., Bin Abdul Salam, Z.A. "A conceptual framework for personalized location-based Services (LBS) tourism mobile application leveraging semantic web to enhance tourism experience", the 3rd IEEE International Conference on Advance Computing (IACC), 2013, pp.287 - 291
- [7] Mohammad AL Nabhan, Suleiman Almasri, Vanja Garaj, Wamadeva Balachandran, and Ziad Hunaiti, "Client-Server Based LBS Architecture: A Novel Positioning Module for Improved Positioning Performance", *International Journal of Handheld Computing Research*, vol.1(3), 2010, pp.1-18
- [8] Ben Niu, Zhengyan Zhang, Xiaoqing Li, and Hui Li "Privacy-area aware dummy generation algorithms for Location-Based Services ", 2014 IEEE International Conference on Communications (ICC), 2014 pp.957 - 962
- [9] K. G. Shin, X. Ju, Z. Chen, and X. Hu, "Privacy protection for users of location-based services," *IEEE Wireless Communications*, vol.19(1), 2012, pp. 30-39.
- [10] Zhu, I., K.-H. Kim, and P. Mohapatra. "An Adaptive Privacy-Preserving Scheme for Location Tracking of a Mobile User." in *IEEE International Conference on Sensing, Communication, and Networking*. New Orleans, USA, 2013, no pp. 9.
- [11] Chen HuangChi, Chen YuJu, Chen ChihYung, Wang Shuming T., Yang JenPin, Hwang ReyChue, "A New Indoor Positioning Technique Based on Neural Network", *Advanced Science Letters*, Vol.19(5), July 2013, pp. 2029-2033
- [12] AL Nabhan, Mohammad Mousa, "Adaptive, reliable, and accurate positioning model for location-based services", *Brunel University School of Engineering and Design PhD Theses*, 2009
- [13] Ray S., Blanco R., Goel, A.K., "Supporting Location-based Services in a Main-Memory Database", 15th IEEE International Conference on Mobile Data Management, 2014, pp.14-18
- [14] Suprio Ray, Rolando Blanco, Anil K. Goel, "Enhanced database support for location-based services", *Proceedings of the 4th ACM SIGSPATIAL International Workshop on GeoStreaming*, 2013, p.22-25.
- [15] Virrantaus, K., Markkula, J., Garmash, A., Terziyan, V., Veijalainen, J., Katanosov, A., Tirri, H., "Developing GIS-supported location-based services," *Proceedings of the Second International Conference on Web Information Systems Engineering*, 2001, vol.2, pp.66-75
- [16] Jerry Gao, X. Bai, W. T. Tsai, and T. Uehara, "Mobile application testing: a tutorial", *IEEE Computer*, vol.47 (2), 2014, pp.26-35
- [17] Ke Zhai, Bo Jiang, Chan, W.K., "Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study," *IEEE Transactions on Services Computing*, 2014, vol.7, no.1, pp.54-67
- [18] Hui-Chun Chua, Gwo-Jen Hwangb, Chin-Chung Tsai, Judy C.R. Tsengc, "A two-tier test approach to developing location-aware mobile learning systems for natural science courses", *Computers & Education*, vol.55(4), 2010, pp.1618-1627
- [19] Bjornestad, S., Tessem, B., Nyre, L., "Design and Evaluation of a Location-Based Mobile News Reader," 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2011, pp.7-10
- [20] Yu J, Tappenden A, Miller J et al., "A scalable testing framework for location-based services," *journal of computer science and technology*, vol.22(2),2009,386-44
- [21] Oum-EI-Kheir Aktouf, Tao Zhang, Jerry Gao, Tadaihiro Uehara, "Testing location-based function services for mobile applications", *The First International Workshop on Mobile Cloud TaaS (MCTaaS 2015)*, 2015, in press.

Generating various contexts from permissions for testing Android applications

Kwangsik Song, Ah-Rim Han, Sehun Jeong, Sungdeok Cha
Department of Computer Science and Engineering
Korea University
Seoul, South Korea
{kwangsik_song, arhan, gifaranga, scha}@korea.ac.kr

Abstract—Context-awareness of mobile applications yields several issues for testing, since the mobile applications should be testable in any environment and with any contextual input. In previous studies of testing for Android applications as event-driven systems, many researchers have focused on using the generated test cases considering only GUI events. However, it is difficult to detect failures in the changes in the context in which applications run. It is important to consider various contexts since the mobile applications adapt and use novel features and sensors of mobile devices. In this paper, we provide the method of systematically generating various executing contexts from permissions. By referring the lists of permissions, the resources that the applications use for running Android applications can be inferred easily. The various contexts of an application can be generated by permuting resource conditions, and the permutations of the contexts are prioritized. We have evaluated the usefulness and effectiveness of our method by showing that our method contributes to detect faults.

Keywords—Android application testing, permissions, various contexts, context-aware application, mobile application testing

I. INTRODUCTION

The proliferation of the novel features and sensors of mobile devices (i.e., operating systems, hardware platforms, and device sensors) has enabled the development of mobile applications that can provide rich, highly-localized, context-aware content to users [1]. In particular, the market for disease diagnostic systems is growing fast due to the development of mobile applications that log personal health data (e.g., blood glucose, blood pressure, and heart rate) by using the sensors, cameras, additional simple adapters (or accessories) in mobile devices and sending the results to the system in real-time. For instance, in the mobile application called *Peek Vision* [2], medical images can be captured by using a clip-on camera adapter that gives high quality images of the back of the eye and can be sent to the system so diagnosis can be done remotely. The mobile application has been designed to be aware of the computing context in which it runs and to adapt and react according to its findings; therefore, it belongs to the category of context-aware applications [3].

The context-awareness of mobile applications yields several issues for testing [4] because the mobile applications should be testable in any environment and with any contextual input [5]. These applications are notified of a change to their context by means of events, and the variability in the running

conditions of a mobile application depends on the possibility of using it in variable contexts. A context represents the overall environment that the application is able to perceive [6]. More precisely, Abowd et al. [7] define a context as: “any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is including the user and applications themselves.”

In previous studies of testing of *Android applications* as event-driven systems, many researchers focused on using the generated test cases considering only GUI events. However, it is difficult to detect failures in the changes in the contexts, which can be influenced by context events, in which applications run. Even in studies that considered context events, the specific event sequences generated based on a limited number of scenarios were considered. This has limitations in terms of finding bugs that occur in various complex contexts. It is important to discover the unacceptable behaviors of an app (such as crashes or freezes) that are often reported in the bug reports of mobile apps and appear when the app is impulsively solicited by contextual events, such as the alerts for the connection/disconnection of a plug (e.g., USB and headphone), an incoming phone call, GPS signal loss, etc. Therefore, for testing mobile applications, we need a systematic testing method to take into account the various conditions in context-aware systems. This is increasingly needed given trends in mobile applications due to the advancement in the novel features and sensors of mobile devices, which reveal new types of bugs [8].

To access resources from Android devices, each Android application includes a manifest file, `AndroidManifest.xml`, which lists the permissions [9] that the application requires for its execution and requests permissions for the resources. By referring to the lists of permissions, the resources that the applications use for running Android applications can be inferred easily. The context events occurred from/by those identified resources, and the state for each condition can be changed by those context events. Thus, we use the permissions to generate the various contexts used for testing Android applications.

To test mobile applications in various contexts, we provide a method for systematically generating **various executing contexts** from permissions. In our paper, an executing context represents a permutation of resource conditions that have variable states, and Graphical User Interface (GUI) event based generated test cases [10] can be run in those contexts. The state of each condition can be changed/sensed/perceived according

to several types of context events, such as:

- events coming from the **external environment** and sensed by device sensors (e.g., Wi-Fi and GPS);
- events generated by the **device hardware** platform (e.g., battery and other external peripheral port, such as USB, headphone, and network receiver/sender); and
- events typical of mobile phones (e.g., the arrival of a phone call or a SMS message).

The brief procedure for generating various executing contexts using permissions is as follows. First, the related resources and their possible states are identified from the permissions. Then, the various executing contexts are generated by permuting the resource conditions that have variable states. Finally, the executing contexts are prioritized and the part of those executing contexts are selected. We applied our testing method to two open-source projects, Open Camera [11] and Subsonic [12]. Experiments reveal that the proposed method is significantly effective in detecting faults.

The rest of this paper is organized as follows: Section II contains a discussion of related studies. Section III explains the definition and the need to use permissions when testing Android applications, and the related resources that can be inferred from the permissions are identified. Section IV explains the procedure to generate various contexts from the permissions. In Section V, we present an experiment to evaluate the proposed approach and discuss the results. We conclude and discuss future research in Section VI.

II. RELATED WORK

Mobile applications are event-driven systems, but, unlike other traditional event-driven software systems, GUI [10], [13]–[15] or web applications [16], they are able to sense and react to a wide range of events. In the following subsections, we discuss the related studies that provide methods for testing *Android applications* as event-driven systems.

A. GUI Testing

Random testing. The UI/Application Exerciser Monkey [13] is part of the Android SDK and generates random user input. Originally designed for stress-testing Android applications, it randomly generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. Monkey testing is a random and automated unit test. The test is not scripted and is run mainly to check whether a system or an application will crash. It is easy to set up and can be used in any application. The cost of using the testing is relatively small. However, detection of only a few bugs is possible.

Model-based testing. *AndroidRipper* [14] is an automated technique implemented in a tool that tests Android applications using a GUI model. *AndroidRipper* is based on a user interface-driven ripper that automatically explores the application’s GUI to exercise the application in a structured manner. More specifically, it dynamically analyses the application’s GUI for obtaining sequences of events that are *fireable* through the GUI widgets. Each sequence provides an executable test case. During its operation, *AndroidRipper* maintains a state

machine model of the GUI (called a GUI Tree). The GUI Tree contains the set of GUI states and state transitions encountered during the ripping process. However, by using generated test cases that consider only GUI events, it is difficult to find failures that could otherwise be detected by considering the changes in the context, which can be influenced by context events, in which applications run.

B. Context-aware Testing

Amalfitano et al. [6] took into account both context events and GUI events for testing Android applications. They manually define reusable *event patterns*—representations of event sequences that abstract meaningful test scenarios. These event-patterns are manually defined after a preliminary analysis is conducted on the bug reports of open source applications. Based on the defined event patterns, test cases are generated using the three scenario-based mobile testing approaches that (1) manually generate test cases, (2) mutate existing test cases, and (3) support the systematic exploration of the behavior of an application (an extension of the GUI ripping technique is presented in [14]). For dynamically recognizing the context events that the application is able to sense and react at a given time, events can be deduced from event handlers. In this work, they also use a set of Intent Messages to figure out the events that are managed by other application components. This set can be obtained by means of static analysis of the a Android manifest file of the application.

The methodology proposed by Amalfitano et al. has some limitations. The number of scenarios that define relevant ways of exercising an application is limited because specific event sequences are considered. By manual analysis by experts, the events possibly trigger a faulty behavior may not be properly identified. By analyzing bug history, a sequence of events that has never occurred might not be chosen, but they may cause catastrophic failures. These event patterns may need to be redefined when testing other types of applications. From the perspective of triggering the context events, the source codes also need to be analyzed and altered. Moreover, the effectiveness of the testing approach is evaluated only by measuring the code coverage. Statement coverage may not be effective and sufficient enough on fault detection capability [17]. In our paper, we provide a systematic method of generating various executing contexts. Since this method may cause to produce many test cases to be run, we provide a prioritization technique to rank the test cases in the order of the likelihood of detecting faults.

III. INFERRING RESOURCES FROM PERMISSIONS

A. Permissions in Android Application

Android uses a system of permissions [9] to control how applications access sensitive devices and data stores. More specifically, to ensure security and privacy, Android uses a permission-based security model to mediate access to sensitive data (e.g., location, phone call logs, contacts, emails, or photos) and potentially dangerous device functionalities (e.g., Internet, GPS, and camera) [18].

To access resources from Android devices, each Android app requests permissions for resources by listing the permissions. Each Android application includes a manifest file,

TABLE I: List of permissions and related resources with their possible states.

Permission	Allows an App to	Related Resources[Possible States]	Android Version
<i>ACCESS_FINE_LOCATION</i>	Access precise location from location sources	Wi-Fi[on off], GPS[on off], Radio[on off]	Android 1.0 ~
<i>INTERNET</i>	Open network sockets	Wi-Fi[on off], Radio[on off]	Android 1.0 ~
<i>CAMERA</i>	Be able to access the camera device	Camera [on off], SD card[free full]	Android 1.0 ~
<i>BLUETOOTH</i>	Connect to paired bluetooth devices	Bluetooth[on off]	Android 1.0 ~
<i>WRITE_CALL_LOG</i>	Write (but not read) the user's contacts data	Radio[on off]	Android 4.0.3 ~
<i>WRITE_EXTERNAL_STORAGE</i>	Write to external storage	SD card[on off]	Android 1.5 ~
<i>BIND_DEVICE_ADMIN</i>	Ensure that only the system can interact with device	Camera [on off], Flash[on off], SD card[free full], Wi-Fi[on off]	Android 2.2.x ~
<i>VIBRATE</i>	Access to the vibrator	Vibrator[on off]	Android 1.0 ~
<i>NFC</i>	Perform I/O operations over NFC	NFC[on off]	Android 2.3 ~
<i>FLASHLIGHT</i>	Access to the flashlight	Flash[on off]	Android 1.0 ~
<i>CHANGE_NETWORK_STATE</i>	Change network connectivity state	Wi-Fi[on off], GPS[on off], Radio[on off]	Android 1.0 ~
<i>CAPTURE_VIDEO_OUTPUT</i>	Capture video output	LCD[on off], Camera [on off]	Android 1.0 ~

AndroidManifest.xml [19], which lists the permissions that the application requires for its execution. When the user wants to install an app, this list of permissions is presented and confirmation is requested. When the user confirms the access, the app will have the requested permissions at all times (until the app is uninstalled). If an application requests the resource without having the appropriate permission, then the Android OS may throw a Security Exception or simply not grant the requested resource [20]. These permission-protected resources are accessed through the Android API and other classes resident on the phone. For example, having the *ACCESS_FINE_LOCATION* permission will give the application access to a number of Android API calls that use resources such as GPS, Wi-Fi, and Radio.

B. Identification of Related Resources from Permissions

We have used the permissions in an app’s manifest file for generating various context used for testing Android applications. By referring to the lists of permissions, the resources that the applications would (potentially) use for running Android applications can be inferred easily, without analyzing source codes of the applications. The context events occur from/by those identified resources, and the state for each condition can be changed by those context events. Thus, by using permissions, we can generate various executing contexts that represent permutations of resource conditions that have variable states.

The latest Android platform release contains a list of 152 permissions. Among them, we focus on the permissions related to communicating with the environments, because they are more critical for making context-aware apps. For each permission, the related resources with their possible states are identified in Table I. It is intuitive to identify the related resources in the permissions of *BLUETOOTH* or *CAMERA*. Meanwhile, in the permission of *ACCESS_FINE_LOCATION*, it covers multiple resources such as GPS, Wi-Fi, and Radio. To consider the variable states of resource conditions, the possible states are defined in terms of an availability (i.e., on or off). It is also worth to note that the table is independent to the features of an app and thus it is reusable.

IV. TESTING ANDROID APPLICATIONS IN VARIOUS CONTEXTS

Fig. 1 represents the overall procedure for generating various executing contexts using permissions.

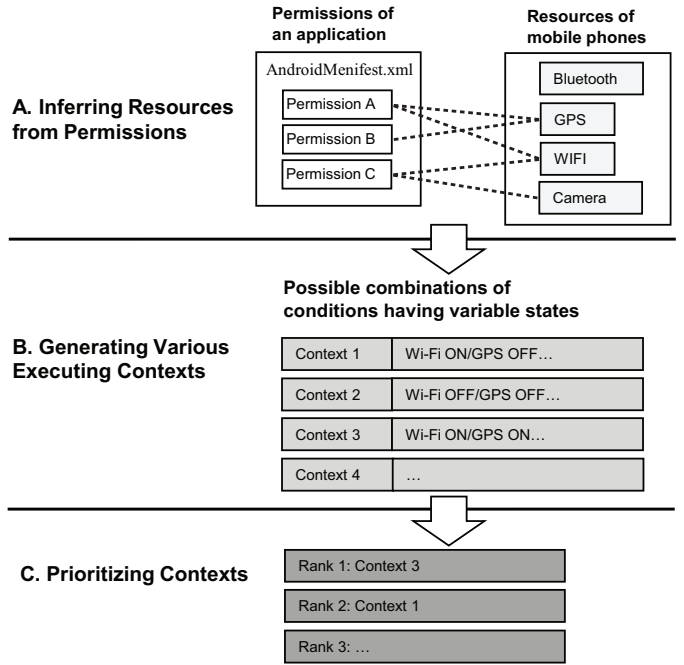


Fig. 1: An overview for generating various contexts after analyzing permissions.

A. Generating Various Executing Contexts

The executing contexts of an app can be generated by permuting resource conditions. For instance, if the resources that an app uses are r_1, r_2, \dots, r_n , and the number of possible states for those corresponding resource conditions are $N(r_1), N(r_2), \dots, N(r_n)$, then the total number of generated executing contexts is $N(r_1) \times N(r_2) \times \dots \times N(r_n)$. For example, if an app’s permission has links with Bluetooth, GPS, and Wi-Fi, then net executing contexts include eight different permutations because each resource condition has two candidate states.

B. Prioritizing Contexts

While the generation of executing contexts is straightforward and easy to automate, the number of generated executing contexts increases as the number of considered resources increase. An app is executed on every test case for all the

generated various contexts, and the test runs increase exponentially. Thus, we need to prioritize the executing contexts to select the contexts to be tested first.

We suggest the two-level prioritizing strategies to rank the generated executing contexts. The first step is weighting each resource condition according to the testing objectives (e.g., testing normal or unacceptable behaviors). To test normal behaviors of the apps, the executing contexts, in which more resources are used, should be more highly ranked. Thus, for example, weights can be assigned to resource conditions as follows: Wi-Fi[on]=1, GPS[on]=1, Camera[on]=1, and SD card[free]=1. If the objective of the testing is to detect unacceptable behaviors of an app, then executing contexts related to the exceptional scenarios should be more highly ranked; thus, the resource conditions constituting those executing contexts need to be weighted, such as Wi-Fi[off]=1, GPS[off]=1, Camera[off]=1, and SD card[full]=1. To obtain the score of each generated executing context, the weights of resource conditions of the executing context are summed.

In the second step, to distinguish the executing contexts that have the same scores, we provide the method to assign weights to individual or combinatorial resources residing in an executing context. We suggest three criteria—frequency, user controllability, and minimum required resource conditions—as follows.

- **Frequency.** It represents how much a resource is required via permissions and is to be used in an app. It counts the identified number of each resource over the lists of permissions. For example, let an app have the permissions in Table I, then the frequency of the Radio resource is four. Thus, frequently identified resources need can be weighted to test more used resource-related behaviors of an app first.
- **User controllability.** It indicates how easily a user can control a resource. For example, users can enable or disable GPS or Wi-Fi but do not control hardware-related sensors directly. Thus, resources that are more user controllable can be weighted to test usable resource-related behaviors of an app first.
- **Minimum required resource conditions.** The certain combinations of resource conditions need to be weighted to test permission-related behaviors first. The rational of the idea comes from the observations that several permissions are related to multiple resources and require minimum resource conditions to provide expected services to an app. For example, the *ACCESS_FINE_LOCATION* permission uses three resources, GPS, Wi-Fi, and Radio; and among the three resources, GPS[on] and Wi-Fi[on] are the *necessary and sufficient* resource conditions to provide the service which is to access precise location from location sources. On the other hand, if we focus on detecting faults, the combination of states that could trigger a faulty behavior (e.g., GPS[on] and Wi-Fi[off]) could be more highly weighted.

V. EVALUATION

We investigated the two research questions in our experiment.

TABLE II: Characteristics for each experimental subject.

Name	Open Camera (Ver. 1.21) [11]	Subsonic for Android (Ver. 4.4) [12]
Description	Taking pictures and providing various features (e.g., zooming, focusing, flashing, and coloring effects)	Playing music and video by receiving media files from the stream server (e.g., personal PC) and supports offline mode and bitrates
Class #	61	265
Method #	399	1038
LOC #	3,790	16,064

TABLE IV: Bugs that can be detected using our approach.

Open Camera		Subsonic	
Fault No.	Bug ID. (refer in [21])	Fault No.	Bug ID. (refer in [22])
1	1	1	150
2	2	2	126
3	9	3	64
4	20	4	102
5	3	5	38
6	11	6	82
7	30	7	46
8	31	8	39
9	37	9	35
10	4	10	32
11	28	11	21
12	33	12	8
		13	4
		14	83

RQ1. Is our testing approach useful for detecting faults?

RQ2. Is our prioritization technique effective in detecting faults?

A. Experimental Design

Two open source projects are chosen as experimental subjects: Open Camera [11] and Subsonic [12]. We selected these as subjects because they are open source projects and their development histories (such as bug issues) are accessible. They contain a relatively large number of classes and methods (large size) as well. Table II summarizes characteristics of each subject.

The testing is performed by running the test cases under each context. In other words, the same test cases had run in an iterative manner as much as the number of the (selected) contexts. We first execute test cases generated from the Android GUI ripper tool [10]. They provide the sequences of events associated with GUI tree paths that link the root node to the leaves of the tree, but the results of statement code coverage on the experimental subjects were low (i.e., average from 45% to 47%). Since GUI-based approaches have limitations for covering all components, we additionally performed testing by focusing on the scenarios that users use more frequently and faulty behaviors may be more occurred.

Table III shows the generated and used executing contexts for Open Camera and Subsonic. As mentioned in Section IV-A, the executing contexts to be tested first need to be prioritized and selected because too many test runs are required, which is computation-intensive. To test the normal scenario, we select the context where all resources are on (active). On the other hand, to test the exceptional scenario, we also select the context where all resources are off (inactive). The contexts that might

TABLE III: Generated and used executing contexts from our approach.

Name	Executing Contexts																																																											
	Permission	Resource[States]	Total #	Used #																																																								
Open Camera [11]	ACCESS_FINE_LOCATION	Wi-Fi[on off], GPS[on off], Radio[on off]	32 = 2 ⁵	<table border="1"> <thead> <tr> <th>Rank</th> <th>Wi-Fi</th> <th>GPS</th> <th>Radio</th> <th>SD card</th> <th>Camera</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>on</td> <td>on</td> <td>on</td> <td>free</td> <td>enable</td> </tr> <tr> <td>2</td> <td>off</td> <td>off</td> <td>off</td> <td>full</td> <td>disable</td> </tr> <tr> <td>3</td> <td>on</td> <td>on</td> <td>on</td> <td>full</td> <td>enable</td> </tr> <tr> <td>4</td> <td>off</td> <td>on</td> <td>off</td> <td>free</td> <td>enable</td> </tr> </tbody> </table>	Rank	Wi-Fi	GPS	Radio	SD card	Camera	1	on	on	on	free	enable	2	off	off	off	full	disable	3	on	on	on	full	enable	4	off	on	off	free	enable																										
	Rank	Wi-Fi		GPS	Radio	SD card	Camera																																																					
	1	on		on	on	free	enable																																																					
2	off	off	off	full	disable																																																							
3	on	on	on	full	enable																																																							
4	off	on	off	free	enable																																																							
CAMERA	Camera [on off], SD card[free full]																																																											
WRITE_EXTERNAL_STORAGE	SD card[free full]																																																											
Subsonic [12]	INTERNET	Wi-Fi[on off], Radio[on off]	128 = 2 ⁷	<table border="1"> <thead> <tr> <th>Rank</th> <th>Wi-Fi</th> <th>Radio</th> <th>Bluetooth</th> <th>SD card</th> <th>Audio</th> <th>MIC</th> <th>CPU</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>on</td> <td>on</td> <td>on</td> <td>free</td> <td>enable</td> <td>on</td> <td>unlock</td> </tr> <tr> <td>2</td> <td>off</td> <td>off</td> <td>off</td> <td>full</td> <td>disable</td> <td>off</td> <td>lock</td> </tr> <tr> <td>3</td> <td>on</td> <td>on</td> <td>off</td> <td>free</td> <td>enable</td> <td>on</td> <td>unlock</td> </tr> <tr> <td>4</td> <td>on</td> <td>on</td> <td>on</td> <td>full</td> <td>enable</td> <td>on</td> <td>unlock</td> </tr> <tr> <td>5</td> <td>on</td> <td>on</td> <td>on</td> <td>free</td> <td>enable</td> <td>on</td> <td>lock</td> </tr> <tr> <td>6</td> <td>off</td> <td>on</td> <td>on</td> <td>free</td> <td>enable</td> <td>on</td> <td>unlock</td> </tr> </tbody> </table>	Rank	Wi-Fi	Radio	Bluetooth	SD card	Audio	MIC	CPU	1	on	on	on	free	enable	on	unlock	2	off	off	off	full	disable	off	lock	3	on	on	off	free	enable	on	unlock	4	on	on	on	full	enable	on	unlock	5	on	on	on	free	enable	on	lock	6	off	on	on	free	enable	on	unlock
	Rank	Wi-Fi		Radio	Bluetooth	SD card	Audio	MIC	CPU																																																			
	1	on		on	on	free	enable	on	unlock																																																			
	2	off		off	off	full	disable	off	lock																																																			
	3	on		on	off	free	enable	on	unlock																																																			
	4	on		on	on	full	enable	on	unlock																																																			
	5	on		on	on	free	enable	on	lock																																																			
	6	off		on	on	free	enable	on	unlock																																																			
	BLUETOOTH	Bluetooth [on off]																																																										
RECORD_AUDIO	Audio[on off], MIC[on off]																																																											
READ_PHONE_STATE	Radio[on off]																																																											
WRITE_EXTERNAL_STORAGE	SD card[free full]																																																											
WAKE_LOCK	CPU[lock unlock]																																																											
MODIFY_AUDIO_SETTINGS	Audio[on off]																																																											
ACCESS_NETWORK_STATE	Wi-Fi[on off], Radio[on off]																																																											
READ_EXTERNAL_STORAGE	SD card[free full]																																																											

cause faulty behavior are also highly ranked. For example, the faults may be more occurred on the situations when SD card is full (it many cause a problem in file processing) and when GPS is on while Wi-Fi is off (it may result in logging wrong location information). When setting these contexts, other resources—not involving these situations—are set to be inactivated. As a result, we selected four among 32 and six among 128 executing contexts in Open Camera and Subsonic, respectively.

Since the subjects are open source projects, we can access the bug histories of Open Camera [21] and Subsonic [22]. We manually analyzed the issues in those repositories and extracted the faults that could have been detected if our testing technique had been used. Then, we execute the test cases in the contexts generated using our approach and discover unacceptable behaviors of the app (such as crashes or freezes). These bugs are matched the corresponding faults extracted from the repositories. For example, we found a crash (i.e., runtime exception: fail to connect camera service) by executing test cases in the context where a camera is disabled. This crash can be matched to the faults of camera malfunctions or exceptions.

To evaluate the effectiveness of our method of prioritizing contexts, we compare three different sequences of contexts in which test cases run: the generated order T, the reversed order T_r, and the prioritized order T_p (which is ranked according to our prioritization method). The order T is the order generated from our approach but not prioritized. The generated order T and the reversed order T_r can be regarded as random sequences.

To quantify the capability of the contexts on fault detection, we use a metric called APFD (Average Percent Fault Detection) [23]. The APFD is calculated by taking the weighted average of the percentage of faults detected over the life of the suite. The higher numbers imply faster (better) fault detection rate. Let T be a test suite containing *n* contexts in which test cases run, and let F be a set of *m* faults revealed by T. Let TF_{*i*} be the fist context in ordering of T' of T which reveals fault *i*. The APFD for test suite T' is given by the equation:

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}.$$

We also measure the fault detection rate according to the order of executing contexts.

B. Results

Number of detected bugs. We found 12 out of 38 bugs in Open Camera, and 14 out of 151 bugs in Subsonic (see Table IV). This results show that our testing is useful in detecting faults.

APFD measure. For Open Camera, # of contexts running with test cases (*n*) is 32, and # of faults (*m*) is 12. The APFD for three orders (i.e., T (generated order), T_r (reversed order), T_p (prioritized order using our approach)) are calculated as follow:

$$T: 0.92 = 1 - (6+1+1+1+1+9+9+3+1+1+1+1)/384,$$

$$T_r: 0.62 = 1 - (3+9+17+17+17+1+1+1+32+17+17+17)/384,$$

$$T_p: 0.97 = 1 - (4+1+1+1+1+2+2+2+1+1+1+1)/384.$$

For Subsonic, # of contexts running with test cases (*n*) is 128, and # of faults (*m*) is 14. The APFD for three orders are calculated as follow:

$$T: 0.96 = 1 - (2+1+1+1+1+2+1+1+6+6+1+1+1+1)/1536,$$

$$T_r: 0.92 = 1 - (10 + 1+1+1+1+10+1+1+5+5+1+1+1+1)/1536,$$

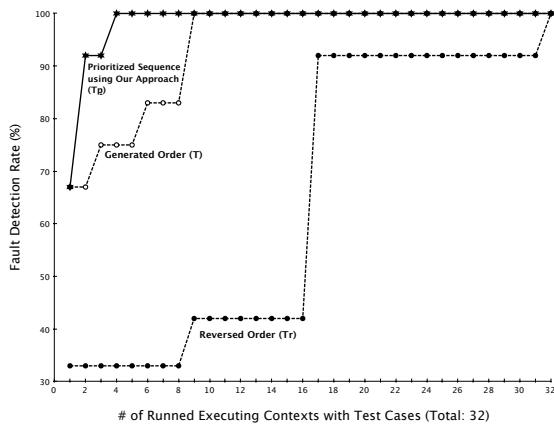
$$T_p: 0.98 = 1 - (1+11+11+11+19+1+11+20+11+11+4+11+3+11)/1536.$$

In both projects, the APFDs for T_p represent the highest scores. Note that, in Subsonic, the number of generated executing contexts is large (i.e., 128) and many of the faults are detected by small number of the executing contexts, thus, the APFD measures are not much different in three orders.

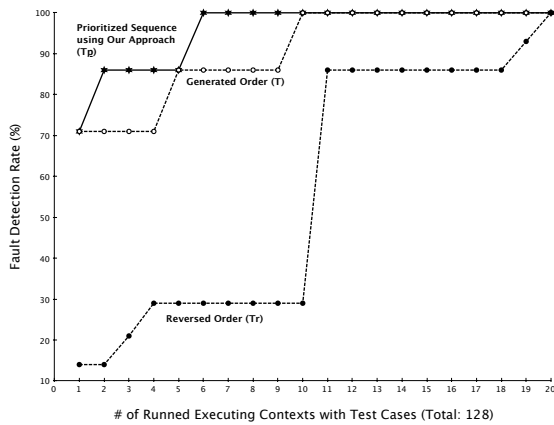
Fault detection rate. We present the graphs of fault detection rate for Open Camera and Subsonic in Fig. 2(a) and in Fig. 2(b), respectively. The graphs show the percentages of faults detected versus the fraction of the contexts used, for each sequence of comparators. For Open Camera, we (T_p) reached to 100% of fault detection rate after running four executing contexts, while the generated order (T) and the reversed order (T_r) required 9 and 32 executing contexts, respectively. For Subsonic, our prioritized sequence (T_p) reached to 100% of fault detection rate after running six executing contexts, while the generated order (T) and the reversed order (T_r) required 10 and 20 executing contexts, respectively. From the results, we can conclude that the order prioritized using our prioritization method results in the earliest detection of the faults.

VI. CONCLUSION AND FUTURE WORK

In our paper, we provide a method for systematically generating various executing contexts from permissions to



(a) Fault detection rate for Open Camera.



(b) Fault detection rate for Subsonic.

Fig. 2: Fault detection rate graphs.

test Android applications. To generate the various contexts, the related resources and their possible states are identified from the permissions. Then, the various executing contexts are generated by permuting resource conditions, and the executing contexts are prioritized and selected. We applied our testing method to two open-source projects and showed the method is effective in fault detection.

For future work, we plan to consider more permissions and identify the relations between the resources and those permissions. We also plan to perform the more detailed experiment for showing the capability of using various contexts. Finally, we plan to devise the method of considering sequences in our contexts for simulating dynamically changing environment.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2014R1A1A2054098). This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2015-H8501-15-1012) supervised by the IITP(Institute for Information & communications Technology Promotion).

REFERENCES

- [1] J. Dehlinger and J. Dixon, "Mobile application software engineering: Challenges and research directions," in *Workshop on Mobile Software Engineering*, 2011.
- [2] Peak Vision, <http://www.peakvision.org/>.
- [3] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.
- [4] D. Amalfitano, A. R. Fasolino, P. Tramontana, and B. Robbins, "Testing android mobile applications: Challenges, strategies, and approaches." *Advances in Computers*, vol. 89, pp. 1–52, 2013.
- [5] H. Muccini, A. Di Francesco, and P. Esposito, "Software testing of mobile applications: Challenges and future research directions," in *Automation of Software Test (AST), 2012 7th International Workshop on*. IEEE, 2012, pp. 29–35.
- [6] D. Amalfitano, A. R. Fasolino, P. Tramontana, and N. Amatucci, "Considering context events in event-based testing of mobile applications," in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 126–133.
- [7] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*. Springer, 1999, pp. 304–307.
- [8] A. Kumar Maji, K. Hao, S. Sultana, and S. Bagchi, "Characterizing failures in mobile oses: A case study with android and symbian," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. IEEE, 2010, pp. 249–258.
- [9] System Permissions, <http://developer.android.com/intl/ko/guide/topics/security/permissions.html>.
- [10] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and G. Imparato, "A toolset for gui testing of android applications," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 650–653.
- [11] Open Camera, <http://opencamera.sourceforge.net>.
- [12] Subsonic, <http://subsonic.org/pages/apps.jsp#android>.
- [13] UI/Application Exerciser Monkey, <http://developer.android.com/tools/help/monkey.html>.
- [14] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using gui ripping for automated testing of android applications," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012, pp. 258–261.
- [15] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "A gui crawling-based technique for android mobile application testing," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 2011, pp. 252–261.
- [16] M. Wang, J. Yuan, H. Miao, and G. Tan, "A static analysis approach for automatic generating test cases for web applications," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 2. IEEE, 2008, pp. 751–754.
- [17] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 435–445.
- [18] X. Wei, L. Gomez, I. Neamtii, and M. Faloutsos, "Permission evolution in the android ecosystem," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 31–40.
- [19] Manifest Permissions, <http://developer.android.com/reference/android/Manifest.permission.html>.
- [20] R. Johnson, Z. Wang, C. Gagnon, and A. Stavrou, "Analysis of android applications' permissions," in *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*. IEEE, 2012, pp. 45–46.
- [21] Open Camera Bug Issues, <http://sourceforge.net/p/opencamera/tickets/>.
- [22] Subsonic Bug Issues, <http://sourceforge.net/p/subsonic/bugs>.
- [23] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *Software Engineering, IEEE Transactions on*, vol. 27, no. 10, pp. 929–948, 2001.

Context-aware Recommendation System with Anonymous User Profile Learning

Yan Liu

School of Software Engineering
Tongji University
Shanghai, China
Email: yanliu.sse@tongji.edu.cn

Yangyang Xu

School of Software Engineering
Tongji University
Shanghai, China

Mei Chen

Decision and System Group
United Technologies
Research Center (China)
Shanghai, China

Abstract—Recommendation system requests huge personal data, including personal information, purchase history, social tag/network, professional/personal preference, etc. Privacy preservation gets more and more concern in modern recommendation system. In this paper, we use surfing data in single session with context-aware learning to generate an anonymous user profile for recommendation, which yields very encouraging results. User profile is generated based on pre-learned hotel profile with pre-assigned weights. Two major behaviors are captured to learn the temporary user profile, which are search and view functions. A novel factor called “irrelevance” is created to measure the sensitivity of user to each item of hotel profile based on the surfing behaviors. A case study on a flight/hotel inquiring and booking website with different application scenarios and results are analyzed.

Keywords—Context awareness; recommendation system; e-service; user profile

I. INTRODUCTION

A recommendation system (RS) is a Web technology that proactively suggests item(s) to user based on side information, which could be user historical records or explicitly stated group preferences. It has been studied for more than ten years in various application areas, including e-commerce, e-health, and social network. Algorithms as content-based filtering [1], [3], [4], collaborative filtering [2], and context-aware prediction are widely applied [5], [7].

The main aim of a recommendation system is to support the website adherence on its user and attract new customers, which might be one of the most critical parameter for modern e-business. Current recommendation system uses user historical data to predict the blanks in the utility matrix (content-based filtering) or historical data of group users to understand potential options based on similar group of people (collaborative filtering). Users might be not preferred to be predicted or do not agree with the prediction, especially when the search varies or the aim is ambiguous. Many papers have discussed the challenge on privacy preservation [7], [8], which raise a big problem for learning algorithm - how can we learn the user profile in an anonymous way without or with limited historical data?

In this paper, we discuss a novel user profile learning method, which conduct recommendation with no user historical data. We utilize temporary user interaction (search

/ view) data, hotel profile, and environment information with a context-aware learning method to understand user’s intention, and develop anonymous but effective user profile for recommendation. This anonymous user profile learning recommendation system is applied in hotel recommendation for a travel booking web site and obtained good results. The article is organized as follows: first, we briefly discuss existing research and challenge on recommendation system. In section III we introduce the booking system and three application scenarios, as well as the major problems and challenges for hotel recommendation. Proposed learning methods and recommendation algorithm is discussed in section IV with results and comparison. Conclusion and future work are in section V.

II. BACKGROUND

Content-based Recommender System focus on properties of items, where the recommendation on items is based on learning the user preference and constraints. It created user specific item profiles (important characteristics of an item) and calculate the similarity of items. It predicts items that user is most likely to be interested in or has highest tendency will accept. Collaborative-filtering RS focus on the relationship between users and items. It measures user similarity for any items to establish a group profile, which recommend items to a user by voting on the group users. Content-based RS needs historical data for single user, and collaborative-filtering RS requires historical data from group users. Both systems require large data for profile learning, and might not work when the utility matrix is sparse.

Context-aware RS attracts more attentions as people realized that taking into consideration on any contextual information, such as time, place, is important. It might be critical to incorporate the contextual information into the recommendation process, especially under certain circumstances (i.e. location related recommendation). Context is a multifaceted concept that has been studied across different research disciplines [5], where RS utilizes the concept from data mining, e-commerce personalization, information retrieval, and other directly related fields. When $R : User \times Items \times Context \rightarrow Rating$, selecting proper item for specific user at set up context environment will generate very

different rating. This is most interesting but challenging part for a context-aware recommendation system.

III. APPLICATION SCENARIO, PROBLEM AND CHALLENGES

In this paper, we develop an effective anonymous hotel recommendation system for a travel booking website who delivers recommendation through email. The recommendation is based on transition information obtained within an interaction session, which can be a flight ticket booking procedure, or an ambiguous hotel inquiry. The main aim is to increase the hotel booking rate and check-in rate by applying certain recommendation system.

This booking website provides flight and hotel inquiring and booking services. It uses email to conduct recommendation, which is quite efficient for following application scenarios:

- New coming users, most of them are unwilling to register or just have a quick search without booking. We ask these users to leave their email address before leaving. We believe that the user who has intent to book a flight/hotel will leave a valid email. We already observed it during daily operation.
- Registered user without log in, which is almost the same situation as the new user until we found the email address was registered. Actually, we found that it made no difference no matter whether the registered user log in or not. It is hard and almost impossible to conduct effective on-line recommendation (very low hit rate make the recommendation annoy). The potential reasons lie in 2 aspects: one is that the historical record is sparse for most user which make the prediction matrix high sparsity with large uncertainty, and large intra-group variance make the recommendations deviate from real intention significantly. Sometime, the users themselves might not have clear target hotels before searching.
- Users inquiring but did not booking would like to receive recommendations especially with promotion, which means the recommendation through email gets attention if it hits the needs truly. This is another observed practice.

A. Scenarios

There are three different application scenario might trigger the recommendation:

- 1) Promotion proposed by hotel, the RS will send email to target users.
- 2) User search flight and finally book one or more itineraries. This means that the user has logged in.
- 3) Anonymous user search flight or hotel information but did not book anything. The email address will be asked before inquiry and the email input is optional.

Scenario (1) and (2), user profile (UP) learning with personal information and historical data is applicable. Price sensitive customers with previous vacation trip(s) are target user for scenario (1). RS sends out emails to potential interested customer, which we call a passive RS. Applicable strategies are:

- select price sensitive users;
- select users having past trips for holiday or vacation within a time range (e.g. 6 months);
- focus on promotion before public holiday.

Scenario (2) and (3) are active RS, where the recommendation is triggered by user actions. For scenario (2), static UP will be created for recommendation based on historical data. A context understanding model will generate a dynamic UP, and the final recommendation will be rated based on hybrid static & dynamic profiles. Scenario (1) & (2) will not be discussed as we are interested in learning UP with no historical data.

B. Problem and Challenges

We believe that scenario (3) is more applicable and preferred by user, as most users do want a good recommendation without leaking too much personal information, especially when RS learns their profiles. We claim that an anonymous UP learned with context in (3) can be sufficient for RS. That is why we choose scenario (3) as our typical case for analysis, and the percentage of scenario (3) is dominant when analyzing the web visits.

In scenario (3), the setup conditions are: a) no historical data, b) the user is anonymous, c) the destination city is known, and d) side information such as viewed flights or viewed hotels is also given. There are two kinds of inquiry behaviors in scenario (3): 3.a) searching the flight itineraries; or 3.b) searching hotels in a city (several cities).

In (3.a), useful inputs are: destination, viewed flight(s), itineraries date/time, and the search date/time. We use *BT* for “Business Trip” and *PT* for “Private Trip” in following analysis. An item profile is created to learn the intention with probability of “Business Trip” versus “Private Trip” based on context understanding:

- Destination and/or any event related to destination (i.e. a commercial show in the destination city) are used to calculate $P(BT|Destination, Events)$ and $P(PT|Destination, Events)$.
- Flights being viewed suggests the acceptable and preferred class and price level. This helps to determine $P(BT|FlightClass, ItineraryTime)$, and $P(PT|FlightClass, ItineraryTime)$.
- Price sensitivity can be inferred from viewed flights if the user viewed several itineraries. The difference between itineraries tells the priority of price vs. time, and the itinerary date tells the flexibility on the trip. $P(BT|PriceSensitivity, TripFlexibility)$,

and $P(PT|PriceSensitivity, TripFlexibility)$ are learned.

- Viewed itineraries time also helps to learn the $P(BT|Distance\ on\ Itineraries\ Time)$, $P(PT|Distance\ on\ Itineraries\ Time)$ as the business trip is more time sensitive than price.
- Searching date/time, esp. date help calculating $P(BT|Weekday/Weekend, Daytime/Nighttime)$, and $P(PT|Weekday/Weekend, Daytime/Nighttime)$.

All the user specific items help to understand the search purpose and determine whether this is a user who might be interested in hotel recommendation (with/without promotion). We found that most user search flight itinerary will make final booking, which suggests that (3.a) can be merged to scenario (2) by learning context-aware item profile.

Scenario (3.b) is the most critical case and will be studied in this paper. An anonymous user profile is developed by combining information from hotel profile (HP) and temporary user interaction data through a context-aware learning schema. Major problems and challenges in (3.b) are as follows:

- 1) High variation and uncertainty on searching content, with several times or dozens times of search.
- 2) There are 1000+ or 2000+ hotels in metropolis or megapolis, such as Beijing, Shanghai, etc.
- 3) Hotel number varies from dozens to thousands in different cities, where the room type and price range vary for same star hotel in different cities.
- 4) City functionality and characteristics have high variance.
- 5) Identify target users from non-target users.
- 6) Learn user profile and understand user intention for accurate recommendation.

For the challenge related to city own functionality, we are not able to tell or utilize the city profile in our model and we will not count this as side information.

IV. SYSTEM, MODEL AND RESULTS

Figure (1) shows the workflow on this anonymous user profile learning system using hotel profile and temporary user data. Hotel profile, including static and dynamic profile, will be updated in a pre-set period. In parallel, the user interaction data will be used to learn the user behaviors and responds according to different hotel profiles. Through the context understanding we can learn the user intention and select target hotel(s) for recommendation.

In (3.b), the designed features for HP based on static data and side information are: 1) GIS/Business Zone; 2) Hotel Star; 3) Price; 4) Facility; 5) Transportation; 6) Rating; 7) Room; 8) Promotion; 9) Event. Static and dynamic HP will be learned from these features, where static feature could be Business Zone, Hotel Star, etc.; and price, promotion are dynamic features.

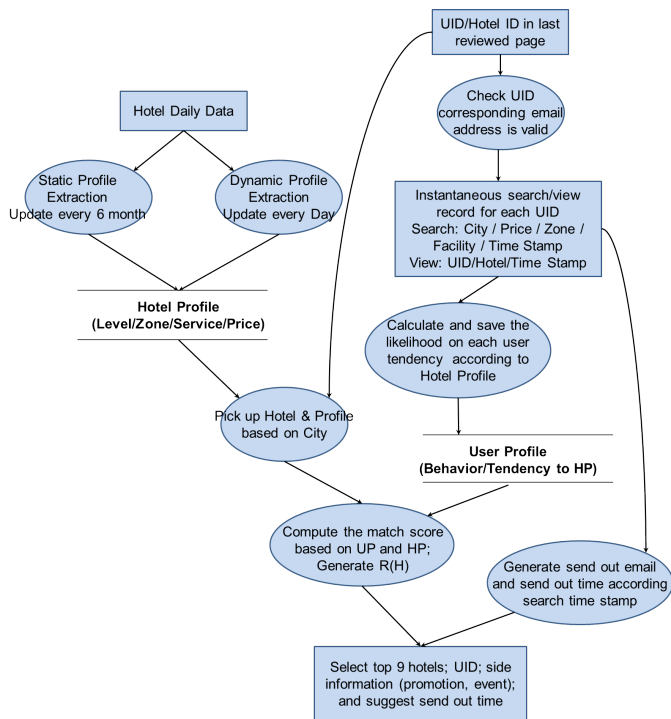


Figure 1. Content-Aware User Profile Learning Recommendation System Architecture.

The inferred UP will be used to answer two questions: 1) whether we are going to send out email with hotels recommendation; 2) what hotels should be recommended. Before creating HP and UP we need to filter out inquires made by agent software such as web crawler. The inquiry with number larger than 100 times contains most random inquires (such as random selected city names), which is very likely to be the scans made by the agent software, and should be removed. This kind of inquiry will be outlier in learning UP and introduce large deviation to real user intention.

A. City and Hotel Profile

We generate HP with different feature sets and weights according to the city. City type also determines how likely we will send out the recommendation to potential user. Table I shows the city category, size, classification rule, features, and methods for HP generation. Category of a city determines which feature set we would like to apply in HP for hotels in that city. Here we use megapolis city as example, which will have all features as we mentioned before.

The HP development is a score calculating and weighting process, where we treat each feature independently. The correlation will be considered during learning the UP based on HP:

- Hotel star is a simple but typical static feature, which

TABLE I
CATEGORY ON HOTEL PROFILE FEATURES ACCORDING TO THE DESTINATION CITY.

CATEGORY RULE	SIZE	CLASSIFICATION	FEATURES	HOTEL PROFILE GENERATION
C_0	MEGAPOLIS	BEIJING, SHANGHAI, SHENZHEN, GUANGZHOU, TIANJIN, CHONGQING	ALL FEATURES	CB + CF
C_1	METROPOLIS	$N_1 < Num_{Hotel} < N_0$	NO GIS/BUSINESS ZONE	CB + CF
C_2	CITY	$N_2 < Num_{Hotel} < N_1$	NO GIS/BUSINESS ZONE/ROOM	RULE-BASED FILTERING
C_3	TOWN	$Num_{Hotel} < N_2$	NO GIS/BUSINESS ZONE/TRANSPORTATION/ ROOM/RATING	RULE-BASED FILTERING
C_4	SPECIAL HOT SPOT	HONGKONG, MACAU, SANYA, OR SEASONAL HOT SPOTS	TRANSPORTATION/ROOM/PROMOTION /EVENT	CB

is grouped into 4 ranges naturally ('two star & below' are put together in a single range). Each hotel can be only assigned in one of the range.

- Price is a typical dynamic feature; and to be simple, we put hard edge on the price range. There are total 10 ranges from 0 to ∞ . Each hotel can have room in multiple price range, and all occupied price range will be marked for a hotel daily.
- Transportation is important but hard to quantize. We use the time to any transportation center as score, where this parameter has less impact than the GIS/Business Zone, especially in metropolis or megapolis city. In our case study, we will not use this feature for HP generation.
- Normalized rating score collected from Hotel Evaluation Website is used directly. This feature has small impact for HP.
- Facility only counts in WiFi, breakfast, parking, which has 0/1 value, corresponding to yes/no.
- Business Zone (BZ) is a unique feature used in this travel booking website, which can be treated as a demographic GIS area. There are about 50 to 100 BZ in metropolis or megapolis city. Each BZ has one unique index number (in each city), where the index does not have numerical meaning and cannot be grouped based on the value.

B. User Profile and Ranking Model

We select user whose inquiry time is 10 - 50 as target user. We define 'behavior' as the operator conducted in the web page. In each successful inquiry, the UP will have accumulated score updated based on the user's 'behavior'. The detailed calculation will be discussed late, and 'behaviors' combined with HP will generate different scores. 'Behaviors' are:

- 1) 'Search': defined as $U_{search}()$. Considered parameters include price range (p), star (s), business zone (z), and facility (f) as $U_{search}(p, s, z, f)$. The parameters and the $U_{search}()$ expressions are:

- $U_{search}(P)$, where $P = [min_{price}, max_{price}]$ is the price range from min to max.
 - $U_{search}(star_2) + U_{search}(star_3)$ if multiple star hotels are selected.
 - $U_{search}(z)$, where zone only has single value.
 - facility has 3 categories, and the parameters can be written as $U_{search}(facility_1) + U_{search}(facility_2) + U_{search}(facility_3)$.
- 2) 'View': defined as $U_{view}()$ with only single parameter 'Hotel' as there is no specific information provide by the user. We will use hotel profile in this parameter.
 - 3) 'Order': contains historical data, which will not be used in UP learning but for validation.
 - 4) 'Count': defines the number of a specific 'behavior' happened in a user. The parameter is $U_{search}()$ and $U_{view}()$.

Users have both view and search record in single inquiry (one event), and the score calculated by $U_{search}()$ will have larger weight. Score tells how far the user is interested in this feature and will be calculated for each feature based on $U_{search}()$ and $U_{view}()$. The calculated individual feature score will be normalized, weighted with user sensitivity on this feature, and then summarized for final ranking. There are 2 different score calculation methods:

- Unique feature for a hotel (i.e. star, zone), the search will have twice counting on 'search' than 'view'. For example, the score for star i : $Star_{i,Score} = 2 \times num(U_{search}(star_i)) + num(U_{view}(Hotel.star = i))$, where $i \in [2, 5]$.
- Features having multiple parameters (i.e. price):
 - 1) Each $U_{search}(p)$, a 10 element vector a (1×10) is created based on $price(minPrice, maxPrice)$. The searched or viewed price ranges falling in the $(minPrice, maxPrice)$ is marked with 1 in corresponding elements in a .
 - 2) Each $U_{view}(Hotel)$, vector b (1×10) is created based on hotel profile, where the price ranges learned from HP is marked with 1 in corresponding elements in b .

3) Price score is the summary of all vectors:
 $U_{pricescore} = a_i + b_j$, $i = 1, \dots, M$, $j = 1, \dots, N$.

- Hotel rating is summarized based on search and view hotels, and normalized. Similar work for transportation.
- Other binary features (promotion and event) take value of 0/1 based on hotel/city daily updated status, meaning Yes/No.

Two weights w_i and w_j are applied to modeling the user true intention upon the calculated feature scores; where w_i represents user sensitivity and w_j represents confidence. We use ‘zone’, ‘star’, and ‘price’ as example to calculate w_i and w_j and explain the main ideas. Let’s assume a megapolis city, which has zone index (1-70), hotel star (2-5), and price range (1-10). A user UID ‘001’ has 20 view records.

- w_i : no previous information on user preference, we assume each feature set is uniform distributed. Use ‘zone’ as example, the sensitivity can be calculated as: $w_{i,zone} = \frac{\sum_{r=1}^R I_{zone}(i,r)}{\sum_{r=1}^R \sum_{k=1}^N I_{zone}(k,rr)}$, where $N = 70$, $R = 20$, and I_{zone} is the indicator function with $I_{zone}(i,r) = 1$ when the i^{th} zone is selected in the r^{th} record.
- w_j : variance level works as confidence.

1) Feature ‘star’ and ‘price’ have numerical meaning on their values. We use inversed L_2 distance between selected parameters as confidence.

$$w_{j,star} = \frac{1}{\sum_{m,n=1}^{20} (star_m - star_n)^2}.$$

2) Feature ‘zone’ has no numerical meaning on its index. We use occurrence vector and standard deviation to model the zone confidence. For example, we have $zone_j$ view record $m_j = \{1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0\}$, where 1/0 means selected/no. Then the standard deviation D_j represents how reliable this user like $zone_j$. Averaged all 70 D_j we can final confidence D as $w_{j,zone}$.

Ranking model is to calculate the likelihood of each hotel that user might be interested. Given

- w_i ,
- w_j ,
- features for destination city ($F_{ct} := \{F_j, j = 1, \dots, n_{ct}\}$);

where n_{ct} is the total number of features. We calculate $R(H) = \sum_{j=1}^{n_{ct}} w_j \times \sum_{i=1}^{N_j} w_i \times HP(F_{ct})$; where $HP(F_{ct})$ is the hotel profile (given city and its specific features).

C. Validation and Results

Performance validations are conducted in two categories:

1) user inquires hotels and make the booking at the same day (noted as $T_{user} \equiv 0$); 2) user inquires hotels, does not make booking instantaneously but book the hotel within

10 days (note as $T_{user} \equiv 1$). We use $T_{user} \equiv 2$ to refer user never make the booking (including user did not input email address). Category (2) is verified with UID and associated email address, where the email address was obtained during inquiry and late booking. Only users used same email address are considered as same user and will be used for validation as $T_{user} \equiv 1$. The comparison is made between the real booked hotel and our recommended top 9 hotels. It means anyone of our recommended 9 hotels is the same hotel as the user booked hotel, we counted as a successful hit.

Table II shows the number of people with 3 situations stated above in a 7 days’ record. A clear pattern of weekday vs. weekend is shown, especially for $T_{user} \equiv 2$. This information will be used in determine sending email time.

TABLE II
USER NUMBER AND DISTRIBUTION.

DAY	TOTAL	$T_{user} \equiv 0$	$T_{user} \equiv 1$	$T_{user} \equiv 2$
1	1396	148	595	653
2	1329	175	591	563
3	1314	128	596	590
4	1311	142	573	596
5	1363	142	628	593
6	942	119	448	375
7	837	84	399	354

Table III shows the hit rate on hotels for 2 validation tests. From this table we find that we have obtained pretty good hit rate in a fully anonymous way, which means this system is valuable. Also, the hit rate of $T_{user} \equiv 0$ is less than $T_{user} \equiv 1$ (almost half) might due to fewer records for $T_{user} \equiv 0$. People make book at the same day always have clear target with less inquiries. Actually, for our RS, $T_{user} \equiv 1$ is our target people and the hit rate is fairly good. The calculated $R(H)$ values also give us a clear boundary on $T_{user} \equiv 1$ and $T_{user} \equiv 2$ people, which is useful to determine whether we need to send out recommendation.

TABLE III
HIT RATE FOR $T_{user} = 0$ AND $T_{user} = 1$.

DAY	$T_{user} \equiv 0$	$T_{user} \equiv 1$
1	21.62%	41.17%
2	28.00%	47.55%
3	26.56%	47.48%
4	26.35%	42.11%
5	23.24%	42.19%
6	26.05%	44.19%
7	26.19%	41.85%
AVERAGE	25.43%	43.65%

V. CONCLUSION AND FUTURE WORK

In this paper, we address several major challenges in context-aware RS. An anonymous user profile-learning

schema allows we provide a recommendation with privacy protection. Target users are separated from others successfully. The final hit rate from validation result indicates that it is feasible to design RS in an anonymous way under some circumstance. We also address challenges in context representation and semi-structure log data analysis, which are very critical in RS. Well-designed architecture ensures the RS work in an efficient way providing real time recommendation.

There are two major concerns for the future work. One is designing sophisticated HMI to understand the user intention (some initial work [9]), to explain the rationale behind recommendation to end-user. The recommendation could be a decision or action. The RS can have high risk in determining what to recommend, especially smart decision support. This requires integration on context awareness, user intention understanding, and recommendation expression as a whole picture. The other is Meta data design for map-reduce structure to handle big data challenge. For a big data flow on line processing system, it is necessary and important to have parallel processing capability. Immigrate this RS to a cloud based structure should be next step.

ACKNOWLEDGMENT

This work was financially supported by the Science and Technology Commission of Shanghai Municipality (12dz1507000).

REFERENCES

- [1] Adomavicius, G., Tuzhilin, A.: Towards the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Trans. on Data and Knowledge Engineering*, 17(6), 734–749 (2005)
- [2] Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1), 5–53 (2004)
- [3] Pazzani, M.J., Billsus, D.: Content-Based Recommendation Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) *The Adaptive Web*. LNCS, vol. 4321, pp. 325–341. Springer-Verlag, Springer (2007)
- [4] Di Noia, T., Mirizzi, R., Claudio Ostuni, V., Romito, D., Zanker, M.: Linked open data to support content-based recommender systems. In: 8th International Conference on Semantic Systems(ACM), New York (2012)
- [5] Adomavicius, G., Tuzhilin, A.: Context-Aware Recommendation Systems. In: *Recommender systems handbook*. pp. 217–253. Springer-Verlag, Springer (2011)
- [6] Wang, S.L., Wu, C.Y.: Application of context-aware and personalized recommendation to implement an adaptive ubiquitous learning system. *Expert Systems with applications*. 38(9), 10831–10838 (2011)
- [7] Verbert, K., Manouselis, N., Ochoa, X., Wolpers, M., Drachsler, H., Bosnic, I., Duval, E.: Context-aware recommender systems for learning: a survey and future challenges. *IEEE Trans. on Learning Technologies*. 5(4), 318–335 (2012)
- [8] Jones, M. T.: Recommender systems, Part 1: Introduction to approaches and algorithms. Technical report, IBM (2013) <http://www.ibm.com/developerworks/library/os-recommender1/>
- [9] Pu, P., Chen, L., Hu, R., Hu: Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*. 22(4-5), 317–355 (2011)

Recommendation in the Digital TV Domain: an Architecture based on Textual Description Analysis

Felipe Ramos

Federal University of Campina Grande
Campina Grande, Brazil
feliperamos@copin.ufcg.edu.br

Alexandre Costa

Federal University of Campina Grande
Campina Grande, Brazil
antonioalexandre@copin.ufcg.edu.br

Reudismam Rolim

Federal University of Campina Grande
Campina Grande, Brazil
reudismam@copin.ufcg.edu.br

Gustavo Soares

Federal University of Campina Grande
Campina Grande, Brazil
gsoares@computacao.ufcg.edu.br

Hyggo Almeida

Federal University of Campina Grande
Campina Grande, Brazil
hyggo@dsc.ufcg.edu.br

Angelo Perkusich

Federal University of Campina Grande
Campina Grande, Brazil
perkusic@dee.ufcg.edu.br

Abstract—Recommendation systems have been used in several application domains, most recently for TV (Digital TV, Smart TV, etc.). Several approaches can be used to recommend items, tags, etc., mainly based on user feedback. However, in the Digital TV domain, user feedback has to be done generally by using the remote control, which should be avoided to improve user experience, since assigning explicit feedback to items is restricted by the characteristics of this domain (difficulties when typing with the remote control, etc.). Moreover, in the Smart TV environment several types of items can be recommended (movies, musics, books, etc.). Thus, the recommendation should be generic enough to suit to different content. To solve the problem of acquiring explicit feedback and still generate personalized recommendations to be used by different Smart TV applications, this work proposes a recommendation architecture based on the extraction and classification of terms by analyzing the textual descriptions of TV programs present on electronic programming guides. In order to validate the proposed solution, a prototype using a real dataset has been developed, showing that using the recommended terms it is possible to generate final recommendations for different Smart TV applications.

Keywords—*Digital TV, Term Classification, Term Recommendation.*

I. INTRODUCTION

Due to the significant growth in recent years of medias such as TV and Internet, the access to information has become increasingly easy. Therefore, a new range of services and information are available for users. However, given the large amount of content available, it is difficult for users to find relevant information [1]. In this context, recommendation systems (RSs) are presented as important tools, because they help users to select items and contents.

Recommendation systems work with the concept of items and users, where “item” is the general term used to denote what is recommended and “user” is the term used to represent who consumes the recommendation [11]. To perform personalized recommendations, in general, RSs need to identify the main

features of users or items (e.g., item descriptions, user histories, user ratings, etc.). These features are used to construct profiles, which are generated from the extraction of dataset information.

RSs can be used in different application domains to help users make better choices [11], [8], especially in Digital TV (DTV), where a large number of channels, programs and the content diversity complicate the user decision making [9]. In DTV, user profiles are usually generated by analyzing their TV viewing history, whereas item profiles are usually generated by extracting information from the electronic programming guide (EPG) [15], which provides program details, such as title, description, categories, etc. Besides the content diversity of Digital TV, with the advent of Smart TVs, which integrate features of the Internet and Web 2.0 on TV devices [3], the range of options for users has become even greater, since they can access information from multiple applications with different types of items. In addition of programs and channels, Smart TVs applications can recommend items of distinct types, such as movies, news, videos, musics, etc. Thus, given the heterogeneity of applications, the Smart TV domain demands an approach that ensures interoperability of recommendations, otherwise recommendation systems have to be developed for applications of different contexts, or at least different components of these RSs must be developed based on item types or features, such as the profile managers, where for each application an user would have a different profile.

Although some previous works focus on recommendation architectures applied to DTV / Smart TV [2], [3], [7], they normally deal with the recommendation of specific items (i.e., TV programs). However, many recommendation approaches can be applied with different features and goals, such as recommendation of terms, tags, etc.

In the recommendation of tags, for example, the model based on user feedback is well known [16], where the user assigns tags to several items, but this model is not suitable for the domain of DTV, since the action of assigning tags and giving explicit feedback in DTV is restricted by user

experience requirements, which demand the use of the remote control as less as possible [2], because watching TV and typing with it at the same time is a time consuming and difficult task. So, ways of acquiring implicit information in DTV need to be investigated, such as the extraction of terms from program descriptions present on the EPG.

Hence, DTV / Smart TV domain presents two important specifications that were not faced together in previous works, all the information must be implicitly collected and the recommendation must be generic enough to suit to different contexts. In our work, we propose a recommendation architecture based on the extraction and classification of terms from TV program descriptions. The main steps of the proposed solution are as follows:

- **Profiles generation:** despite the great number of different types of application users can interact, they consume the same type of item, TV programs. In our work, we build user profiles based on their TV viewing histories (i.e., programs they watched before), hence, the users have a unique profile independent of the applications they interact;
- **Term extraction and classification:** to represent the items (i.e., TV programs) we extract terms from program descriptions. In order to generate a generic recommendation, we classify extracted terms based on EPG categories, hence, it is possible to identify terms related to a given application, for example, terms of sports;
- **Recommendation:** instead of recommending programs only, we recommend classified terms based on program recommendation. Our main goal is to generate an intermediary recommendation, thus, it is possible to recommend different items, since the final recommendation for different applications can be processed from the term recommendation, avoiding the need to develop different RSs to different applications.

In order to validate the proposed solution, we developed a prototype using a real dataset, recommending two types of items, movies and books. The prototype consisted in the generation of two recommendation adapters, where the final recommendation was processed from the term recommendation. It showed that is possible to recommend for different Smart TV applications based on terms classified by EPG categories.

II. RELATED WORK

Similar to other areas, Digital TV suffers from information overload due to the growth in the number of TV programs and channels. Therefore, some studies are focused on this application domain [2], [3], [7], [12].

Chang et al. [3] proposed a TV program recommender framework for Smart TV, addressing several issues (such as accuracy, diversity, novelty, etc.), which contains three components: TV program content analysis module, user profile analysis module and user preference learning module.

Bambini et al. [2] described the integration of a recommendation system into FastWeb, a large IP Television

(IPTV) provider. The recommendation system implemented both collaborative and content-based techniques, in order to recommend programs and videos on demand.

Krauss et al. [7] proposed a system (TV Predictor) that includes recommendation mechanisms to Smart TVs, aiming to generate personalized program guides, which consist of personal channels for each user. Additionally, the TV Predictor Autopilot enables the TV set to automatically change the currently viewed channel, allowing the user to watch the personalized programming without further user input.

Unlike previously mentioned works, which intended to recommend specific items (i.e., TV programs), in this work the main goal is to propose an architecture to recommend terms that can be used by different Smart TV applications. The terms are extracted from textual descriptions of programs and classified based on program categories specified on the EPG. Thus, additionally to the program recommendation (component of the proposed architecture), which is processed by the analysis of user viewing histories, the recommendation of terms is also generated. Therefore, the proposed architecture aims to make the process of generating recommendations for applications of different contexts easier, and this is the main difference among this work and others mentioned before.

As a large part of content available is presented in textual format [13] (EPG, for example), some works focus on text categorization [13], [16].

Rossi et al. [13] proposed a textual document categorization algorithm to define a model inspired on a bipartite heterogeneous network. The network consists of two different types of objects: documents and terms extracted from their textual descriptions, in which the training set has some previously classified documents, and the induction consists of assigning weights to terms related to the known document classes. In our work, we adapt the proposed approach to perform the term classification phase, considering each program as a document and program categories as bipartite network classes, as illustrated in Figure 1. Each program on the EPG has its predefined categories, so the adaptation aimed only to identify the relationship between terms extracted from program descriptions and categories.

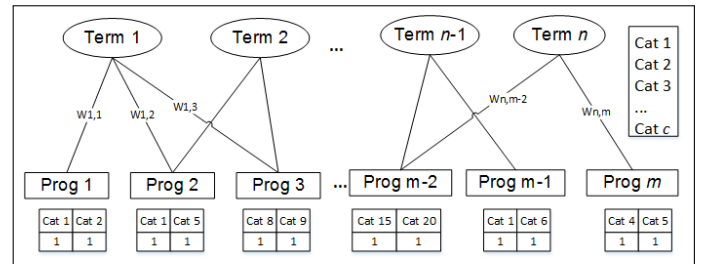


Fig. 1: Bipartite heterogeneous network.

III. PROPOSED SOLUTION

In this section, we present the proposed recommendation architecture, that aims to generate term recommendations by following Digital TV domain specifications to maximize the user experience while watching TV, which are as follows:

- **Different applications:** several kinds of Smart TV applications can be added to the Digital TV domain, featuring different types of items such as books, movies, news, etc. Thus, the generated recommendation must be intermediary, i.e., it must be possible to generate a final recommendation from it, ensuring interoperability;
- **Remote control as interaction source:** user interacts with TV through the remote control. Therefore, to avoid the use of this device and maximize user experience while watching TV, information must be collected by implicit feedback;
- **Implicit feedback:** Digital TV implicit feedback calculation is different from other domains (e-commerce, etc.), since there is a period that a user can consume an item. For example, if a program is presented once a week, a given user can only watch it on the exact day and moment of its transmission. Thus, we calculate a user implicit feedback by dividing the number of times a program was watched by the number of times it was presented;
- **EPG with predefined categories:** unlike other domains, items in Digital TV (TV programs) generally present two categories specified on the EPG. Therefore, with program descriptions and their respective categories it is possible to identify the relationship between terms extracted from descriptions and categories (action, sports, news, etc.). So, it is possible to recommend terms of a specific category related to an application, i.e., to personalize term recommendation by categories (action terms, news terms, sport terms, etc.).

An overview of the proposed architecture is shown in Figure 2, with the following components:

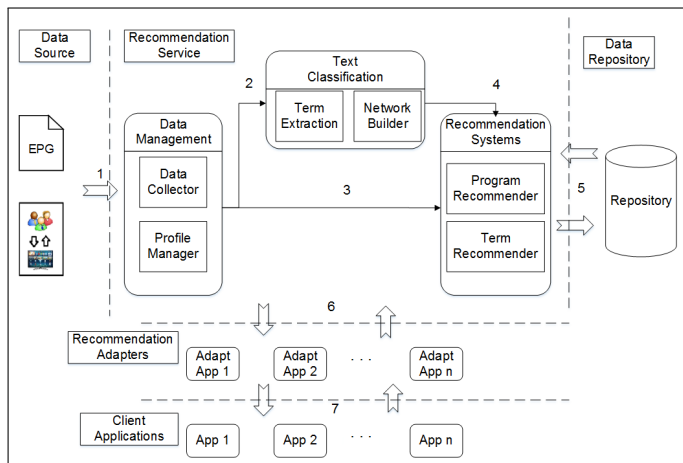


Fig. 2: General architecture.

- **Data Management:** collect (1) and manage information to generate user and item profiles (3 and 2);
- **Text Classification:** classify terms extracted from textual descriptions present on the EPG based on program categories;

- **Recommendation Systems:** generate program and term recommendations based on user and item profiles (3 and 2) and classified terms (4);
- **Recommendation Adapters:** generate final recommendation (7) from the term recommendation (6);
- **Repository:** store and retrieve (5) information from user and item profiles.

A. Data Collection and Management

The data collection of TV programs and users is performed by the Data Collector (Figure 2), which is a common element of recommendation architectures [2]. In the proposed architecture, program information is extracted from the EPG, while user information is collected implicitly by the analysis of their TV viewing history.

The Profile Manager (Figure 2) is responsible for managing the information collected from the data sources (1) and aims to create user and item profiles that are stored in the repository (5) and, subsequently, used during classification and recommendation phases.

Program profiles are formed by their textual features extracted from the EPG (e.g., title, description, categories, etc.). On the other hand, to create user profiles we calculate the implicit feedback by counting the number of times a user u viewed a particular program p and how often this program is weekly presented, and apply Equation 1.

$$\bar{r}_{up} = \left\lfloor \frac{v_{up}}{f_p} \right\rfloor \times 5, f_p > 0, \quad (1)$$

where v_{up} is the number of times the user u watched the program p and f_p represents the number of times the program p is weekly presented. The obtained rating is a normalized value from 1 to 5. An example of applying Equation 1 is shown in Table I, for example, user 1 watched program 1 three times in a week, and the program is presented three times in the same period. Thus, the user implicit feedback is 5.

TABLE I: Implicit feedback examples

User Id	Prog Id	Prog freq	week	User watch freq	Impl rating
1	1	3	3	5	5
1	2	1	1	5	5
2	1	3	1	2	2
⋮	⋮	⋮	⋮	⋮	⋮
2	3	7	5	4	4

User profile information (3) is used by the Recommendation Systems component (Figure 2) to generate recommendations. On the other hand, item profile information (2 and 3) is used by the Recommendation Systems and by the Text Classification component (Figure 2), which processes term extraction and classification.

B. Term Extraction and Classification

This step consists of extracting terms from TV program descriptions (performed by the Term Extraction component (Figure 2)) and classifying them based on program categories (performed by the Network Builder component (Figure 2)).

The term extraction is performed by mining TV program textual descriptions to retrieve representative terms. To accomplish this task, we first discard *stop words* (less significant words such as prepositions, articles, etc.) [4]. Then, we perform *stemming* (reduce words to roots) [4]. At the end, each program will have a vector of terms where each position in this vector corresponds to the frequency of the term on the program textual description.

The term classification consists of categorizing previously extracted terms into EPG categories. Since each program on the EPG has usually two predefined categories, it is possible to identify the relation between term and category through their co-occurrence. At the end of this phase, each term will have a weight assigned to the categories.

To perform the term classification, we construct a bipartite heterogeneous network, which consists of a network $G = (V, E, W)$ with different types of objects V , a set of connections between objects E — no link between objects of the same type is needed — and a set of connection weights W [13].

Our bipartite network is an adaptation of Rossi et al. [13] proposed one, including two types of objects: terms and programs. The term weight set is given by the matrix $W = \{w_1^T \dots w_\alpha^T\}^T$, where α is the number of terms extracted from program descriptions and w_{ij} is the weight of the term i to the category j . The matrix W has dimension $\alpha \times \phi$ where ϕ is the number of categories. The TV program categories are represented by the vector $c = \{c_1, \dots, c_{|C|}\}$. The terms extracted from program descriptions are represented by the vector $f = \{f_1, \dots, f_\alpha\}$. Each object of the program type has a weight vector for the categories, which is represented by the matrix $Y = \{y_1^T, \dots, y_\theta^T\}^T$, where θ is the number of programs available, and y_{kj} receives the value 1 if the program k has the category j , 0 otherwise. The weight of the relation between programs and terms is given by the matrix $D = \{d_1^T, \dots, d_\theta^T\}^T$, each position d_{ki} represents the frequency of a term i in the description of a program k . The matrix D has dimension $\theta \times \alpha$.

The goal of this classification step is to construct the matrix W . To accomplish this task, we use the IMBHN algorithm [13], which allows inferring the influences of each term for program categories. The IMBHN algorithm performs the process by minimizing the cost function given by Equation 2 [13]:

$$\begin{aligned} Q(W) &= \frac{1}{2} \left(\sum_{j=1}^w \sum_{k=1}^{\theta} (class(\sum_{i=1}^{\alpha} d_{ki}w_{ij}) - y_{kj})^2 \right) \\ &= \frac{1}{2} \left(\sum_{j=1}^w \sum_{k=1}^{\theta} error_{kj}^2 \right), \end{aligned} \quad (2)$$

where,

$$class\left(\sum_{i=1}^{\alpha} d_{ki}w_{ij}\right) = \begin{cases} 1 & c_j = \underset{c_{j^*} \in c}{\operatorname{argmax}}\left(\sum_{i=1}^{\alpha} d_{ki}w_{ij^*}\right) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The algorithm aims to minimize the squared error between the predicted and real values of the program categories. Gradient descent (Least-Mean-Square [13]) adjusts matrix W until a minimum error or a maximum number of iterations is reached (i.e., algorithm stop conditions¹).

C. Term Recommendation

Before generating the term recommendation, we analyze user and item profiles to process program recommendation, in order to identify appropriate items for users, this task is performed by the Program Recommender component (Figure 2).

In this work, we generate lists of recommended programs for users by using a hybrid recommendation system, which is generally a combination of collaborative filtering and content-based approaches, however, any recommendation techniques can be applied. For collaborative filtering we used Matrix Factorization² (MF), in which learning is performed by stochastic gradient descent [6]. Additionally, we analyze user profiles and include into MF recommended lists, programs with ratings greater or equal to 3 in user histories.

Finally, we generate term recommendation through the Term Recommender component (Figure 2). As each recommended program has a vector of terms extracted from its descriptions (Term Extraction (Figure 2)) and classified by EPG categories (Network Builder (Figure 2)), the weight of a recommended term t for a user u is given by:

$$\bar{r}_{tu} = \frac{1}{|P_u|} \sum_{p=1}^{|P_u|} f_{tp} \times r_{pu}, |P_u| > 0, \quad (4)$$

where $|P_u|$ is the number of recommended programs containing the term t to the user u , and f_{tp} is the number of occurrences of the term t on program p description and r_{pu} is the recommended rating to the program p for the user u . Thus, terms more frequent in recommended program descriptions tend to receive a higher weight. Using this weight, an application can use recommended terms according to its own requirements.

D. Recommendation Adapters

Recommendation Adapters (Figure 2) are responsible for generating the final recommendation (7), which is used by the Client Applications (Figure 2). Thus, for each application a corresponding adapter must be created, which uses (6) the Recommendation Service (Figure 4.2).

¹maximum number of iterations = 1000, minimum error = 0.01.

²We use MyMediaLite Recommender System Library - <http://www.mymedialite.net/>.

We create a recommendation adapter based on categories related to the corresponding application and its list of users (7), which are reported to the recommendation service (6), and then the lists of recommended terms to users are returned (6), such as, terms of action, comedy, sports, etc.

A possible technique to generate final recommendation is to compute the similarity between vectors of recommended terms and vectors of extracted terms from textual information of recommendable items (e.g., movies, books, news, etc.). Here, we use the cosine similarity [14], which compute the angle between two vectors (common terms) x and y of size m .

An illustration of the final recommendation generation is shown in Figure 3, where we want to select the most suitable drama movie to the user interest among three possible choices (i.e., Movie 1,2,3). The first step is to obtain the user recommended terms of drama (i.e., user feature vector for the drama category), then, we extract terms from movie descriptions, in order to create each movie vector of terms (i.e., movie feature vectors). Finally, we calculated the cosine similarity among the user feature vector and each movie feature vector. Thus, based on the calculated similarity, Movie 3 is the best related to the user interest, since it achieved the greatest similarity, which means that Movie 3 feature vector and the user feature vector present a greater number of common terms.

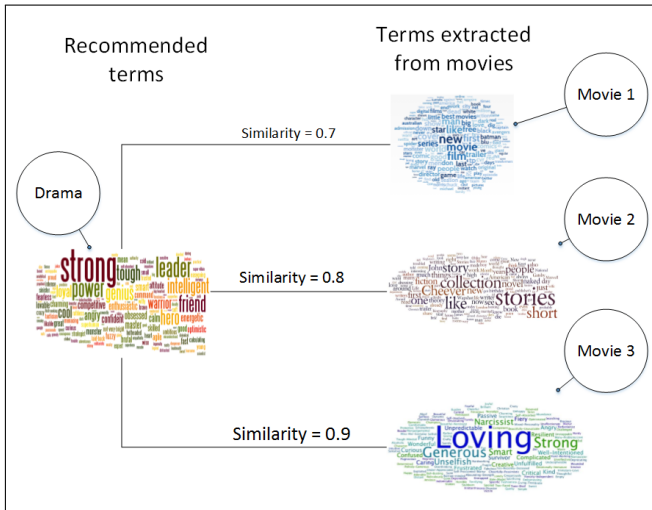


Fig. 3: Example of final recommendation generation.

IV. VALIDATION

In order to validate our solution we used a real dataset, which consists of information about TV users and their viewing histories. To collect the data we conduct a survey, where the participants filled a form³ with information about the programs they usually watch and their corresponding frequency view during a week. The dataset is composed of 63 users, 112 programs and 29 types of program categories.

The main goal of the validation is to proof the interoperability of recommendations, i.e., recommend different items from the term recommendation. Thus, we developed a prototype, which consists in creating two recommendation adapters

to recommend for the 63 users two different items, books and movies. The proposed architecture should be integrated into a DTV architecture. However, for matters of validation the process took place in a desktop environment.

To generate the final recommendation (books and movies), we create two recommendation adapters, one for each type of item, where the following steps were performed:

- 1) We specified EPG categories related to the book and movie applications (e.g., action, adventure, comedy, drama, police and thriller);
- 2) We determined the list of users of each application, here we considered the 63 participants of the survey;
- 3) Then, we processed term recommendation by category, i.e., for each user we got recommended term of action, adventure, comedy, etc.;
- 4) After obtaining the recommended terms, we generated the final recommendation of books and movies. The process was performed as follows: first, we mined item descriptions to extract representative vectors of terms. Then, we calculated the similarity (i.e., cosine similarity) between recommended terms by category for each user and extracted term vectors, recommending items with greater similarity.

The proposed approach can be used to recommend different types of items (such as videos, news, products, etc.), since they have textual description and related categories. Therefore, the same way the movie and book recommendations were processed it could be done for other types of items.

In order to evaluate our solution, we compare it with a non-personalized approach (i.e., without classification and recommendation), in which the user feature vector consists of the occurrence of terms extracted from user histories, i.e., the weight assigned to each term here is given by its number of occurrences in program descriptions. The main research question we want to answer is the following:

P1 - The use of our personalized approach (PA) improves the precision of the final recommendation compared to the non-personalized one (NP)?

To answer that, we formulate the following hypotheses:

H_0 : The precision of PA is greater than NP for category c and item i .

Where c is a category of type action, adventure, comedy, drama, police or thriller, and i is an item of type movie or book. So that, we have 12 hypotheses.

Since the assumption of normality was not met based on Shapiro-Wilk test, we used the non-parametric test of Wilcoxon signed-rank (95% confidence, i.e., $\alpha = 0.05$).

In Table II can be seen the results of movie recommendation precisions, i.e., the mean of the precisions ($P@5^4$) calculated for each user, and the p-value related to the hypothesis tests, where we conclude that our solution outperformed the NP approach for the categories action, police and thriller. For the remainder, the two approaches are statistically equal.

⁴Given a category, how many of the top 5 recommended movies are of that same category?

³<http://goo.gl/kEDKWt>

TABLE II: Movie recommendation precisions

	Categories					
	Act.	Adv.	Com.	Dra.	Pol.	Thr.
PA	66%	44%	34%	23%	44%	38%
NP	50%	45%	34%	26%	20%	30%
p-value	1e-06	0.44	0.22	0.82	4e-11	7e-06

In Table III can be seen the results of book recommendation precisions, i.e., the mean of the precisions ($P@4^5$) calculated for each user, and the p-value related to the hypothesis tests, where we conclude that our solution outperformed the NP approach for all studied categories, except for drama.

TABLE III: Book recommendation precisions

	Categories					
	Act.	Adv.	Com.	Dra.	Pol.	Thr.
PA	47%	33%	20%	7%	44%	74%
NP	42%	25%	14%	29%	23%	38%
p-value	0.04	0.001	0.01	1	1e-09	8e-11

Finally, we can conclude that the use of our solution of extraction, classification and recommendation of terms achieved better results for both, movie and book recommendations.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a recommendation architecture based on the extraction and classification of terms from TV program descriptions, applied to Digital TV domain. The proposed approach for the term extraction is important to overcome problems arising from the interaction constraints between users and TV. Another significant contribution is the term classification and recommendation approach, which allows different applications to use the generated recommendations, which is not possible in approaches that recommend items.

The prototype showed the feasibility of the proposed solution, ensuring that is possible to recommend different items using the term recommendation approach. Thus, to generate the final recommendation to a given application, it is only necessary to create a corresponding recommendation adapter, which uses the term recommendation based on the categories from the EPG closely related to the application. So that, only the recommended terms that have higher weight for these categories are returned.

As future work, a study about different ways to extract terms will be carried out. Some works have focused on the extraction of product attributes [5], [17]. The proposed approaches in these studies can be evaluated and integrated to the current proposal with the aim to obtain a better representation of TV programs, and hence, to get more significant recommendations.

As the solution proposed in this paper is based on the extraction of EPG content, which in some cases may contain reduced information [10], a possible future work is to investigate ways to obtain data from different sources to enrich the EPG information, and improve the textual representation of TV programs, for example, using information from Wikipedia⁶ [10].

⁵Given a category, how many of the top 4 recommended books are of that same category?

⁶<http://www.wikipedia.org/>

ACKNOWLEDGMENT

The authors would like to thank CAPES for support this work.

REFERENCES

- [1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *TKDE, IEEE*, 17(6):734–749, 2005.
- [2] R. Bambini, P. Cremonesi, and R. Turrin. A recommender system for an iptv service provider: a real large-scale production environment. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 299–331. Springer, 2011.
- [3] N. Chang, M. Irvan, and T. Terano. A tv program recommender framework. *Procedia Computer Science*, 22(0):561 – 570, 2013. KES2013.
- [4] A. Fariña, N. R. Brisaboa, G. Navarro, F. Claude, A. S. Places, and E. Rodríguez. Word-based self-indexes for natural language text. *ACM Trans. Inf. Syst.*, 30(1):1:1–1:34, Mar. 2012.
- [5] R. Ghani, K. Probst, Y. Liu, M. Krema, and A. Fano. Text mining for product attribute extraction. *SIGKDD Explor. Newsl.*, 8(1):41–48, June 2006.
- [6] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.
- [7] C. Krauss, L. George, and S. Arbanowski. Tv predictor: Personalized program recommendations to be displayed on smarttvs. In *Proceedings of the 2Nd International Workshop on BigMine, BigMine '13*, pages 63–70, New York, NY, USA, 2013. ACM.
- [8] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.
- [9] A. Martinez, J. Pazos Arias, A. Vilas, J. Duque, and M. Nores. What's on tv tonight? an efficient and effective personalized recommender system of tv programs. *Consumer Electronics, IEEE Transactions on*, 55(1):286–294, 2009.
- [10] C. Musto, F. Narducci, P. Lops, G. Semeraro, M. Gemmis, M. Barbieri, J. Korst, V. Pronk, and R. Clout. Enhanced semantic tv-show representation for personalized electronic program guides. In J. Masthoff, B. Mobasher, M. Desmarais, and R. Nkambou, editors, *User Modeling, Adaptation, and Personalization*, volume 7379 of *Lecture Notes in Computer Science*, pages 188–199. Springer Berlin Heidelberg, 2012.
- [11] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 1–35. Springer, 2011.
- [12] R. Rolim, F. Barbosa, A. Costa, G. Calheiros, H. Almeida, A. Perkusich, and A. Martins. A recommendation approach for digital tv systems based on multimodal features. In *Proceedings of the 29th Annual ACM SAC, SAC '14*, pages 289–291, New York, NY, USA, 2014. ACM.
- [13] R. Rossi, T. de Paulo Faleiros, A. de Andrade Lopes, and S. Rezende. Inductive model generation for text categorization using a bipartite heterogeneous network. In *12th ICDM, 2012*, pages 1086–1091, 2012.
- [14] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th WWW, WWW '01*, pages 285–295, New York, NY, USA, 2001. ACM.
- [15] X. Shi and J. Hua. An adaptive preference learning method for future personalized tv. In *KIMAS'05, 2005. International Conference on*, pages 260–264, 2005.
- [16] Y. Song, L. Zhang, and C. L. Giles. Automatic tag recommendation algorithms for social recommender systems. *ACM Trans. Web*, 5(1):4:1–4:31, Feb. 2011.
- [17] T.-L. Wong, W. Lam, and T.-S. Wong. An unsupervised framework for extracting and normalizing product attributes from multiple web sites. In *31st annual international ACM SIGIR, SIGIR '08*, pages 35–42, New York, NY, USA, 2008. ACM.

A Collaborative Method to Reduce the Running Time and Accelerate the k-Nearest Neighbors Search

Alexandre Costa

Federal University of Campina Grande
Campina Grande, Brazil
antonioalexandre@copin.ufcg.edu.br

Reudismam Rolim

Federal University of Campina Grande
Campina Grande, Brazil
reudismam@copin.ufcg.edu.br

Felipe Barbosa

Federal University of Campina Grande
Campina Grande, Brazil
feliperamos@copin.ufcg.edu.br

Gustavo Soares

Federal University of Campina Grande
Campina Grande, Brazil
gsoares@computacao.ufcg.edu.br

Hyggo Almeida

Federal University of Campina Grande
Campina Grande, Brazil
hyggo@dsc.ufcg.edu.br

Angelo Perkusich

Federal University of Campina Grande
Campina Grande, Brazil
perkusic@dee.ufcg.edu.br

Abstract—Recommendation systems are software tools and techniques that provide customized content to users. The collaborative filtering is one of the most prominent approaches in the recommendation area. Among the collaborative algorithms, one of the most popular is the k-Nearest Neighbors (kNN) which is an instance-based learning method. The kNN generates recommendations based on the ratings of the most similar users (nearest neighbors) to the target one. Despite being quite effective, the algorithm performance drops while running on large datasets. We propose a method, called Restricted Space kNN that is based on the restriction of the neighbors search space through a fast and efficient heuristic. The heuristic builds the new search space from the most active users. As a result, we found that using only 15% of the original search space the proposed method generated recommendations almost as accurate as the standard kNN, but with almost 58% less running time.

Keywords—knn, recommendation systems, collaborative filtering, space restriction.

I. INTRODUCTION

The emergence of Web 2.0 brought a significant increase in the volume of information available on the Internet, contributing to the information overload problem [10], that overwhelm the user with useless (in most cases) choices. Hence, recommendation systems (RS) [15], [1], [17] gained prominence and became very attractive for both the production sector and academic field. These systems bring together computational techniques in order to provide custom items (movies, music, books, etc.) to users, thus facilitating their choices. In the recommendation area, a prominent approaches is the collaborative filtering (CF) [22] that recommends items based on ratings of users with common interests to the target user. The state-of-the-art in recommendation field is formed by latent factor models [18], where some of the most successful implementations are based on Matrix Factorization (MF) [12]. In its basic form, the MF characterizes users and items with vectors of latent factors inferred from the pattern of the rated items. The latent factors represent aspects of physical reality, but we can not specify which aspects are these, therefore it is impossible to justify the provided recommendation.

Considered state-of-the-art, before the emergence of latent factor models, the k-Nearest-Neighbors (kNN) [11], [20], [21] is an instance-based learning algorithm. In the recommendation area this method is widely used as a collaborative technique for rating prediction. In contrast to latent factor techniques, the kNN recommendations can be justified, since they are generated from the nearest neighbors data. The main limitation of kNN is that its performance is inversely proportional to the size of the dataset. As the number of users and items grows, the computational cost to apply the method rises quickly, which decreases its time performance.

In this paper, we propose a collaborative method to reduce the running time and accelerate the nearest neighbor search, called Restricted Space kNN (RS kNN). This method restricts the search space to a percentage p of the original one, using a heuristic based on selecting the most active users. As a result, we found that with only 15% of the original space it is possible to generate high accuracy recommendations, but with almost 58% less running time.

This paper is organized as follows: In Section II, we present a background of the recommendation area. In Section III, we describe the related work. In Section IV, the proposed method is presented. Section V contains a description of the experiment conducted and its results. Section 6 refers to the conclusion of the work.

II. BACKGROUND

In this Section, we present basic concepts of the recommendation area, that will allow to understand the proposed method.

A. Collaborative Filtering

Collaborative Filtering is one of the most recurring approaches in the recommendation area. Its techniques recommend items based on the ratings of other users. The ratings can be implicit, when they are measured based on user's behavior (e.g, viewing time, number of hits, etc.), or explicit, when users clearly express their interest on items through numerical grades, for example. The idea behind CF is that users with similar rating pattern tend to rate new items in a

similar manner. In CF, an instance is represented by a user feature vector that records which items were evaluated and which not. An advantage of collaborative techniques is the object representation independence, since CF techniques use only user ratings, which enables to work even with items in which the content extraction can be complex, such as audio and video. Another advantage refers to the recommendations diversity, since CF can suggest items different from those the user showed interest in the past.

Collaborative methods can be grouped as memory-based or model-based algorithms. In memory-based algorithms the dataset is loaded at the moment in which the recommendations are generated. They are easier to implement and can better adapt to changes of user interests. In contrast, model-based algorithms generate recommendations using a model previously constructed from the dataset. They can provide more accurate recommendations, but the model construction is an expensive step.

A common scenario in collaborative filtering is the rating prediction, where a user rating to a given item is inferred. In this scenario, it is assumed that items with higher values are more interesting. Ratings can be represented in different scales, usually in 1-5 stars. The quality of the prediction is normally measured through error-based metrics, calculated from the difference between the predicted value and the real user rating. A common metric for this purpose is the Mean Absolute Error (MAE) represented by Equation 1, where $r(u, i)$ is the rating of a user u to an item i , $r'(u, i)$ corresponds to the prediction generated for u about i and $|I_u|$ is the size of the item set evaluated by u .

$$MAE = \frac{1}{|I_u|} \cdot \sum_{i \in I_u} |r'(u, i) - r(u, i)| \quad (1)$$

B. k -Nearest Neighbors

The kNN is a memory-based method, thus it is necessary to have all the training set stored to do the recommendation process. In larger datasets the kNN computational cost can grow quickly. This occurs because as the number of users and items grows the kNN demands more memory to store the data, more similarity calculations and more time to perform the neighbors selection (because the search space becomes larger).

Recommendations are based on the k nearest neighbors ratings, where a similarity measure to select the nearest neighbors must be defined. This measure has a direct impact on the kNN results, because it is used to determine how close two users are. The similarity between two users is calculated from the items they have rated simultaneously. A popular similarity measure in the recommendation field is the Pearson correlation, represented by the Equation 2, where $|I_{au}|$ is the size of the item set simultaneously evaluated by users u and a , $x = r(u, i)$ and $y = r(a, i)$. The Equation 2 differs from the traditional form, because it is adapted to perform faster calculations.

$$sim(a, u) = \frac{|I_{au}| \sum_{i \in I_{au}} xy - \sum_{i \in I_{au}} x \cdot \sum_{i \in I_{au}} y}{\sqrt{[|I_{au}| \sum_{i \in I_{au}} x^2 - (\sum_{i \in I_{au}} x)^2] \cdot [|I_{au}| \sum_{i \in I_{au}} y^2 - (\sum_{i \in I_{au}} y)^2]}} \quad (2)$$

The recommendation process consists in making a prediction for the set of items not evaluated by the user. One of the most common ways to accomplish this goal is through the Equation 3, where U_a is the set of nearest neighbors of the target user a and $\overline{r(a)}$ corresponds to the average ratings of the user a .

$$r'(a, i) = \overline{r(a)} + \frac{\sum_{u \in U_a} sim(a, u) \cdot (r(u, i) - \overline{r(u)})}{\sum_{u \in U_a} sim(a, u)} \quad (3)$$

III. RELATED WORK

Boumaza and Brun [3] proposed a method based on the restriction of the nearest neighbors search space. In traditional search, it is necessary to check the similarity among the target user with each other user in the dataset, and then select the k nearest neighbors. On large datasets, this task becomes expensive. In their work, Boumaza and Brun used a stochastic search algorithm, called Iterated Local Search (ILS) [13]. The ILS returns a subset of users, Global Neighbors (GN), in which the neighbors are chosen. The idea is to accelerate the neighborhood formation by seeking the neighbors in a smaller subset, rather than search in the entire dataset. As a result, the proposed method can reduce the search space to 16% of the original while maintaining the accuracy of recommendations near to those achieved by traditional search. We adapt their work by introducing a new heuristic to perform a faster and less expensive GN selection, instead of using the ILS algorithm.

Friedman et al. [5] proposed an optimization to k -Dimensional Tree (kd tree). Originally proposed by Bentley [2], the kd tree is an algorithm for storing data to be retrieved by associative searches. The kd tree structure provides an efficient search mechanism to examine only those points closer to the point of interest, which can reduce the nearest neighbors search time from $O(N)$ to $O(\log N)$. In Friedman's work, the goal was to minimize the number of points examined during the search, that resulted in a faster search. Gother et al. [8] developed a method which represents a slight variation of the work done by Friedman et al. [5]. In the kd tree construction each node is divided into two, using the median of the dimensions with greater variance between the points in the subtree, and so on. As a result, the method got 25% faster in the classification step. The kd tree based methods differ from ours, because they accelerate the neighbor selection using a non-linear search by partitioning the whole space into non-overlapping regions.

Other works use the approximate nearest neighbors (ANN) concept to deal with the kNN search problem. The ANN methods do not guarantee to return the exact nearest neighbors in every case, but in the other hand, it improves speed or memory savings. An algorithm that supports the ANN search is locality-sensitive hashing (LSH). According to Haghani et al. [9], the main idea behind the LSH is to map, with high probability, similar objects in the same hash bucket. Nearby objects are more likely to have the same hash value than those further away. Indexing is performed from hash functions and from the construction of several hash tables to increase the probability of collision between the nearest points. Gionis et al. [7] develop a LSH method to improve the neighbors search for objects represented by the points of dimension d in a Hamming space $\{0, 1\}^d$. Their method was able to overcome in terms of speed the space partitioning tree methods, when data are stored on disk. Datar et al. [4] proposed

a new version of the LSH algorithm that deals directly with points in a Euclidean space. To evaluate the proposed solution experiments were carried out with synthetic datasets, sparse vectors with high dimension ($20 \leq d \leq 500$). As a result, they obtained performance of up to 40 times faster than kd tree. The ANN methods are related to ours, because they aim to accelerate the neighbor search by giving up accuracy in exchange for time reduction.

IV. RESTRICTED SPACE kNN

In standard kNN, the nearest neighbors are selected by checking the similarity of the target user with each other user in the dataset. For every user, a distinct neighborhood has to be formed. When the number of users and items in the training set grows, the kNN running time decreases, because its computational cost rises quickly. To minimize this problem we proposed a collaborative method derived from the k-Nearest Neighbors for rating prediction. Our method is based on the restriction of the neighbor search space. In Figure 1, we can see the main idea of the proposed method in contrast to the standard approach. The search space is reduced to a percentage p of the original one, which is accomplished by selecting a subset of users capable to offer ratings to generate accurate recommendations. Then, the neighbor search is performed in the new space, which allows it to be faster, since the space becomes smaller. Finally, the most similar users to the target one are chosen to form his neighborhood.

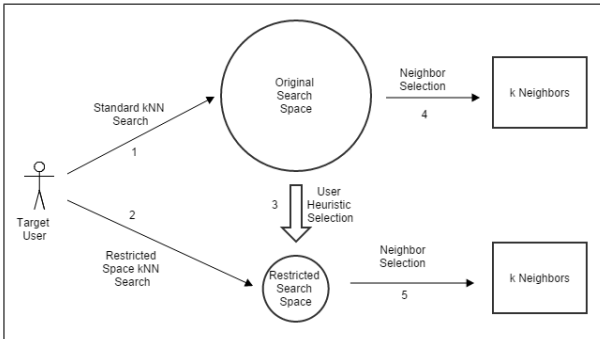


Fig. 1: Standard search and restricted Search

The RS kNN was inspired by the work presented by Boumaza and Brun [3]. Their method was able to achieve accurate recommendations with a reduced search space that corresponds to 16% of the original one. A stochastic search algorithm called Iterated Local Search does the user selection. The ILS is an efficient algorithm for optimization problems, but requires a considerable running time and expensive operations to achieve its results. Given this limitation, we saw an opportunity to improve Boumaza and Brun’s work [3]. Therefore, instead of using the ILS algorithm, we propose a faster and accurate heuristic to select the percentage p of users who will compose the new search space.

Our approach aims to reduce computational cost and improve running time, but it is susceptible to lose recommendation accuracy, since it works with a considerably smaller amount of data. In order to minimize such loss, it is necessary to define an efficient heuristic for the user selection. In our work, we investigated the following heuristics¹:

- **Similarity Average:** users are selected according to their similarity average. For each user, we calculate his similarity with each other that remains in the dataset and then get the average. Those with the highest averages are chosen;
- **Number of Connections:** select users according to their ability to be neighbors. Each time a user shows a positive

similarity with another he receives one point. In the end, those with the highest scores are selected;

- **Neighbors Recurring:** users are scored according to the number of times that arise between the k nearest neighbors of a target user. For example, we check the k nearest neighbors of a target user and then assign one point for each neighbor. This process is repeated with all users in the dataset and at the end, we have the list of the most common neighbors;
- **Most Active Users:** it selects users according to the number of items rated. Those with the largest quantities are chosen;
- **Distinct Items:** it corresponds to a variation of the previous heuristic, with the aim of bringing together users with the greatest possible number of distinct items. It selects users that, together, offers the most distinct set of items.

V. EXPERIMENT

We conducted an experiment to evaluate the proposed method. The experiment was divided into two stages. The first aims to evaluate the heuristics described in Section IV, in order to choose the most suitable to compose our method. The second corresponds to the evaluation of the proposed method by comparing it with implementations developed to accelerate the kNN search. In the experiment, we focused on measuring time performance (in seconds) and accuracy (error rate of the predictions, measured by the Mean Absolute Error metric).

The experiment was executed on a machine with Core i7 2300K (3.4 GHz) processor, 8GB of DDR3 RAM and Windows 7 64-bit. We used the Java language and Eclipse² (Kepler) in its 64-bit version for developers. We also used two external libraries, the MyMediaLite [6], which is specialized in recommendation systems methods, and the Java Statistical Analysis Tool (JSAT) [16], which offers dozens of machine learning algorithms.

A. Dataset

We choose two popular datasets that contain real data from movie domain. The first one is the MovieLens 100K, which has 943 users, 1,682 movies and 100,000 ratings. The second has 6,040 users, 3,952 movies and 1,000,209 ratings. Both have ratings on a scale of 1 to 5 stars that represent the level of the user interest to an item.

The data were segmented following the 10-fold cross-validation pattern, which consists in the separation of 90% of the data for training and 10% for testing. This process was repeated five times to provide a final amount of 50 samples for execution.

B. Setting Parameters

Before going on with the experiment we had to set some parameters. Thus, we used the standard kNN algorithm for rating prediction, whose implementation was based on the source-code available in MyMediaLite library. We performed 10 execution on the MovieLens 100K dataset, focusing in accuracy. The parameters and their values are listed below:

- **Similarity measure:** we compared the Pearson correlation with another popular measure in the recommendation area, the Cosine similarity [19]. As we can see in Table I, the Pearson correlation provides a lower MAE, which means more accurate recommendations. Therefore, we chose the Pearson correlation as similarity measure for our approach.

¹The Similarity Average and Number of Connections were already introduced in [3]. The others were proposed in our work.

²www.eclipse.org

- **Number of nearest Neighbors (k):** choosing the optimal value for k is not a trivial task, because it can vary according to the data. In Figure 2, we can see that for the MovieLens 100K the best k is 30, because it provided the lowest error rate. We used the same k for the larger dataset to maintain the speed gains. In addition, a greater k would increase the running time.
- **Percentage of the search space (p):** we used the Most Active Users heuristic to find the optimal value for p . A greater p would give better MAE, but at the expense of the running time. Thus, our choice was given by a trade-off between the MAE and the running time. According to the Figure 3 when p reaches 15%, it seems to be the best trade-off. Besides it, we thought that should be important to choose a p closer to the one in Boumaza and Brun [3], since our work is inspired by theirs.

TABLE I: Comparison of Pearson Correlation and Cosine Similarity

Similarity measure	MAE
Pearson	0.6813
Cosine	0.7100

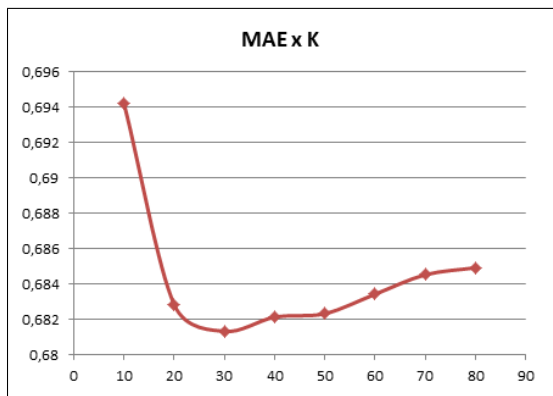


Fig. 2: Variation of k and its respective error rates

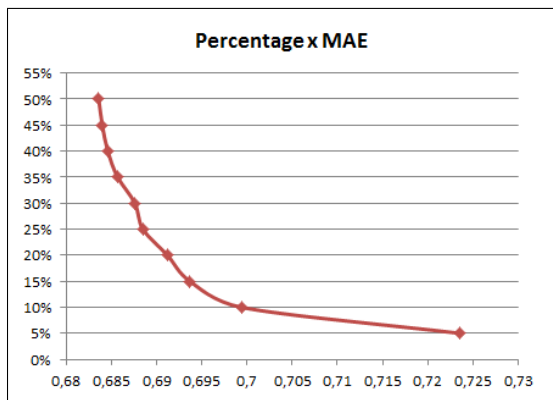


Fig. 3: Variation of p and its respective error rates

C. Heuristic Evaluation

The heuristic evaluation process was based on the time needed to select the users to compose the new search space and the accuracy that they can provide. Tables II and III contain the results of each heuristic. Similarity calculations are expensive, thus the heuristics *Similarity Average*, *Number of Connections* and *Neighbors Recurring* demanded the largest time to select their users. The others showed significantly

shorter times, because they use less expensive operations, since there is no similarity computing. Regarding the accuracy, the results were more homogeneous, there was even a decrease in error rate in the larger dataset.

The results showed the heuristic *Most Active Users* as the best option to compose the proposed method. It provided the smallest error rate, resulting in more accurate recommendations. In addition, the selection time was the best among the heuristics and remained almost constant in the tests with the larger dataset.

TABLE II: Results of the heuristic evaluation on MovieLens 100K

Heuristic	Time (s)	Error (MAE)
Most Active Users	0.001	0.6937
Distinct Items	0.072	0.6954
Similarity Average	1.604	0.7053
Number of Connections	1.591	0.7046
Neighbors Recurring	1.631	0.6994

TABLE III: Results of the heuristic evaluation on MovieLens 1M

Heuristic	Time (s)	Error (MAE)
Most Active Users	0.002	0.6546
Distinct Items	1.097	0.6549
Similarity Average	92.713	0.6588
Number of Connections	92.652	0.6579
Neighbors Recurring	100.390	0.6553

D. Method Evaluation

To evaluate our method, we chose four approaches that correspond to implementations derived from the kNN method. They were developed to accelerate the nearest neighbor selection. Their performance were measured under accuracy and running time and the results are presented in Tables IV and V. The compared approaches are:

- **k-dimensional tree (kd tree):** implemented in JSAT tool. Corresponds to the traditional form [2];
- **Locality Sensitive Hash (LSH):** implemented in JSAT tool. It was based on the algorithm presented by Dating et al. [4];
- **Standard k-Nearest Neighbors (kNN):** corresponds to the kNN for rating prediction. We implemented it based on the source code of the MyMediaLite library;
- **Iterated Local Search (ILS) kNN:** we implemented this method based on the paper presented by Boumaza and Brun [3].

Three of them (standard kNN, ILS and the RS kNN) are collaborative techniques from the recommendation field that were developed focusing on the rating prediction task. They usually deal with sparse vectors and try to “fill” each missing value of the vectors. Regarding the running time, our method achieved the best values, being almost 58% faster (in MovieLens 1M) than the standard kNN, which took second place. The running time of ILS was much greater than the others, because its fitness function needs to check the errors provided by the subsets of users build at each iteration of the algorithm. Furthermore, as a non-deterministic method, it is impossible to predict the number of iterations needed to find the final subset. As expected, the accuracy of the recommendations generated by our method was a little lower than standard kNN, but considering the running time gain, the results were very promising.

The kd tree and LSH methods were originally implemented for the classification task. They generally work with dense vectors and

aim to label an unknown instance. In the proper domain, they are capable to reduce the search time from linear to logarithmic level, although in this experiment this behavior was not evidenced. The classification methods presented poor performances in running time and accuracy. The kd tree tends to lose great performance with high dimension vectors ($d \geq 10$) [7], [5], a problem known as the curse of dimensionality [14], which contributes to the low performance, since the datasets are composed of high dimension sparse vectors. The LSH prioritizes time instead of accuracy, since it returns the approximated nearest neighbors. This reason justifies the high error rate of the LSH. The running time performance of the LSH was unexpected, because this algorithm is designed to be fast even with high dimension data. We believe the data sparsity was responsible, because it reduces the probability of collisions, making difficult to group users in the same hash bucket and consequently it increases the search time.

TABLE IV: Results on MovieLens 100K

Method	Running Time (s)	MAE
kd tree	22.24	0.8333
LSH	11.13	0.7528
ILS	1233.15	0.7083
kNN	4.35	0.6826
RS kNN	2.45	0.6937

TABLE V: Results on MovieLens 1M

Method	Running Time (s)	MAE
kd tree	1576.01	0,7875
LSH	454.42	0,7014
ILS	7892.7	0,6591
kNN	242.13	0,6500
RS kNN	100.87	0,6546

VI. CONCLUSION

In this paper, we presented the Restricted Space kNN, a collaborative method to reduce the running time and accelerate the k-Nearest Neighbors search. The RS kNN focuses on the idea of restricting the nearest neighbors search space using a fast and efficient heuristic for user selection. The method was capable to perform up to 58% faster than standard kNN. The Most Active Users heuristic was quick in reducing the search space, getting together a set of users able to provide accurate recommendations. Using only 15% of the original search space we have achieved an error rate just 0.7% higher than the standard method.

The main limitation of our method refers to the validation process. We evaluated it in only one domain, which makes difficult to generalize the results achieved. Our method showed great performance gains in exchange for a small reduction in accuracy, however, we cannot guarantee that the error rate will remain constant in other domains.

As future work, we intend to investigate the effects of the proposed method in a larger dataset, because we noticed that the accuracy gap between our method and the standard kNN became smaller when the data increased. In the MovieLens 100K, we obtained an absolute difference of 0.0111, whereas with MovieLens 1M, the difference was reduced to 0.0046. In addition, we also intend to investigate the proposed method in the item predicting scenario with implicit feedback.

REFERENCES

- [1] A. Azaria, A. Hassidim, S. Kraus, A. Eshkol, O. Weintraub, and I. Netanel. Movie recommender system for profit maximization. In *Proceedings of the 7th ACM Conference on Recommender Systems, RecSys '13*, pages 121–128, New York, USA, 2013. ACM.
- [2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.
- [3] A. Boumaza and A. Brun. Stochastic search for global neighbors selection in collaborative filtering. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12*, pages 232–237, New York, NY, USA, 2012. ACM.
- [4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04*, pages 253–262, New York, USA, 2004. ACM.
- [5] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, Sept. 1977.
- [6] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the Fifth ACM Conference on Recommender Systems, RecSys '11*, pages 305–308, New York, USA, 2011. ACM.
- [7] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases, VLDB '99*, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
- [8] P. Grother, G. T. Candela, and J. L. Blue. Fast implementations of nearest neighbor classifiers. *Pattern Recognition*, 30(3):459–465, 1997.
- [9] P. C.-M. K. A. P. Haghani. Lsh at large – distributed knn search in high dimensions. 2008.
- [10] R. Janssen and H. de Poot. Information overload: Why some people seem to suffer more than others. In *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles, NordiCHI '06*, pages 397–400, New York, USA, 2006. ACM.
- [11] L. Jiang, Z. Cai, D. Wang, and S. Jiang. Survey of improving k-nearest-neighbor for classification. In *Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery - Volume 01, FSKD '07*, pages 679–683, Washington, USA, 2007. IEEE Computer Society.
- [12] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.
- [13] H. R. Lourenco, O. C. Martin, and T. Stutzle. Iterated local search. In *Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science*, pages 321–353. Kluwer Academic Publishers, 2002.
- [14] F. Murtagh, J.-L. Starck, and M. W. Berry. Overcoming the curse of dimensionality in clustering by means of the wavelet transform. *The Computer Journal*, 43(2):107–120, 2000.
- [15] D. H. Park, H. K. Kim, I. Y. Choi, and J. K. Kim. A literature review and classification of recommender systems research. *Expert Syst. Appl.*, 39(11):10059–10072, Sept. 2012.
- [16] E. Raff. Java Statistical Analysis Tool. <https://code.google.com/p/java-statistical-analysis-tool/>, Sept. 2013.
- [17] R. Rolim, F. Barbosa, A. Costa, G. Calheiros, H. Almeida, A. Perkusch, and A. Martins. A recommendation approach for digital tv systems based on multimodal features. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC '14*, pages 289–291, New York, NY, USA, 2014. ACM.
- [18] M. Rossetti, F. Stella, and M. Zanker. Towards explaining latent factors with topic models in collaborative recommender systems. In *Database and Expert Systems Applications (DEXA), 2013 24th International Workshop on*, pages 162–167, Aug 2013.
- [19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web, WWW '01*, pages 285–295, New York, USA, 2001. ACM.
- [20] B. Wang, Q. Liao, and C. Zhang. Weight based knn recommender system. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2013 5th International Conference on*, volume 2, pages 449–452, Aug 2013.
- [21] L. Xiong, Y. Xiang, Q. Zhang, and L. Lin. A novel nearest neighborhood algorithm for recommender systems. In *Intelligent Systems (GCIS), 2012 Third Global Congress on*, pages 156–159, Nov 2012.
- [22] J. Zhou and T. Luo. Towards an introduction to collaborative filtering. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 4, pages 576–581, Aug 2009.

Achieving Efficient Access Control via XACML Policy in Cloud Computing

Xin Pei, Huiqun Yu, Guisheng Fan
Department of Computer Science and Engineering
East China University of Science and Technology
Shanghai 200237, China
Email: yhq@ecust.edu.cn

Abstract—One primary challenge of applying access control methods in cloud computing is to ensure data security while supporting access efficiency, particularly when adopting multiple access control policies. Many existing works attempt to propose suitable frameworks and schemes to solve the problems, however, these proposals only satisfy specified use cases. In this paper, we take XACML as the policy language and build up a logical model. Based on this, we introduce the fine-grained data fragment algorithm to optimize the policies, whose *resource* property represents physical meaningful data blocks. Data are organized in a tree structure, where each leaf node represents a minimal physical meaningful data block, and internal nodes are combined data types. This method can eliminate conflicts and redundancies among rules and policies, thus to refine the policy set and achieve fine-grained access control. Our approach can also be applied to processing multi-types of data, and experiments are carried out to show the improvements of efficiencies.

Keywords—Access control; Policy optimization; Data fragment; XACML; cloud computing

I. INTRODUCTION

In the last several years, cloud computing brings us great convenience on data outsourcing by providing nearly unlimited storage resources on demand [1]. This allows content providers to create, manage, and control the personal data remotely with high efficiency. Moreover, the charge manner of cloud service is pay-as-you-use which costs relatively lower prices compared with self-maintenance. As promising as it is, cloud storage service also involves many challenges [2], such as the problems of fine-grained access control on multiple data types and the confidentiality of private data. The traditional access control methods are only applicable to rigid data objects, and the policy decision on a request results in either *permit* or *deny*.

Motivated by the requirements of high performance and flexible access control, XACML (eXtensible Access Control Mark-up Language) [3] is proposed to solve data access problem in cloud computing. XACML is an XML-based language, and it contains a hierarchical logic model which is applied to a particular decision request in access control policies for Web applications and Web services. Meanwhile, XACML offers a large set of built-in functions, data types, combining algorithms, and standard profiles for defining application-specific features. There are lots of prior works on applying XACML as access control policy, which focus on policy attesting, conflict detection and policy optimization, etc.

Mont and Pearson propose the ‘sticky policy’ based on XACML to facilitate access control for outsourced data [4] [5], and Trabelsi extends this policy to the cloud environment [6]. However, their proposals only focus on the system framework and the shared data is considered in a single type. Hu and Ahn introduce a description logic (DL)-based policy management approach for Web access control policies, they adopt Answer Set Programming (ASP) to formulate XACML [7], and they further propose a method for conflict detection and resolution in [8]. However, DL cannot fully cover XACML semantics, and it fails to handle complex comparisons, multi-type of decisions as well as combining algorithms. Wang and Feng propose a rule redundant elimination method based on related types of hierarchical attributes tree and provide an XACML policy optimization engine [9], but the access efficiency depends on the amount of rules. Said and Shehab propose a framework for policy evaluation [10], and Lin and Rao suggest a similarity measurement technique among policies [11]. Meanwhile, Bertolino and Daoudagh propose an automated testing method for XACML [12]. Their works are worth well in policy attesting and measurement, but they do not achieve a practical scheme for optimizing policy with consideration of fine-grained access control. Many practical models of XACML are built in [13] [14] [15], but these models are not considerate in decision efficiency, especially for large scale of policies.

Compared with existing policy models, XACML is more comprehensive and intuitive for applying to cloud access control. In this paper, we analyze the logic model of XACML by taking account of all its components and internal functions. Based on this model, we propose the data fragmentation and policy refinement algorithms via building up a three-layers resource access tree, so as to achieve fine-grained access control over multi-types of outsourced data. In the end, we discuss a case study on healthcare records management, and the performances are illustrated by experiments using XACML tools.

II. XACML ANALYSIS

XACML is standardized by the Organization for the Advancement of Structured Information Standards (OASIS) in 2003. In XACML, the complete policy applicable to a particular decision might be composed of a number of individual rules, policies and policy sets, in which there exists a target expression as the criteria for incoming requests, and

all these elements are organized in a hierarchical order [16]. To render an authorization, it must be possible to combine multi-rules to form the single decision applied to the request, and the final decision is either made by the rule ‘effect’ or the combined decisions of children rules and policies.

A. XACML elements

We define the main elements *policy set*, *policy*, *rule*, *target*, *request*, *effect* and *combining algorithm* (CA for short) of XACML syntax as follows, where the values ‘PO’, ‘DO’, ‘FA’, ‘OOA’ represent for ‘permit-override’, ‘deny-override’, ‘first-applicable’ and ‘only-one-applicable’, respectively.

$Polycyset ::= \langle target \rangle, \{ \langle Polycyset \rangle | \langle Policy \rangle \}^+, \langle CA \rangle$

$Policy ::= \langle target \rangle, \{ \langle rule \rangle \}^+, \langle CA \rangle$

$rule ::= \langle target \rangle, \langle effect \rangle, \langle condition \rangle, \langle obligation \rangle$

$target ::= \{ \langle attr_type \rangle, \langle attr_value \rangle, \langle match_id \rangle \}^*$

$request ::= \{ \langle attr_type \rangle, \langle attr_value \rangle \}^+$

$effect ::= 'permit' | 'deny'$

$CA ::= 'PO' | 'DO' | 'FA' | 'OOA'$

Additionally, XACML extends the decision values by appending two extra status ‘not-applicable’ and ‘indeterminate’, based on the previous policy languages with only ‘permit’ and ‘deny’. The status ‘not-applicable’ represents that the request does not match any rule in the designated policy, while ‘indeterminate’ indicates errors in the matching procedure (e.g., The request does not match a ‘critical’ attribute in the target).

B. Decision principles

We describe the principles of XACML to make a decision. On receiving a request, the policy decision point (PDP) executes *target matching* and results *MR* in the set $V_{MR} = \{ 'T', 'F', 'IN' \}$, representing ‘match’, ‘un-match’ and ‘indeterminate’ respectively. If this matching procedure happens in a rule, it leads to a decision according to the rule ‘effect’ and *MR*. Otherwise, if the *target matching* belongs to a policy or policy set, the final decision is integrated by the combining algorithm affecting on children rules and policies.

We denote a user request as a vector $req = \{ a_1, a_2, \dots, a_n \}$, each a_i in req is an attribute value which belongs to the attribute type A_i pre-defined in XACML. The principles are listed as below.

1) *Target matching*. The connection of elements in a target can be either ‘AllOf’ or ‘AnyOf’, which indicates the operations of AND | OR.

Assume a request matches with K elements in the target, formally, $req(K) : A_1 \times A_2 \times \dots \times A_K \rightarrow V_{MR}$. The ‘AllOf’ property performs as in (1), and the ‘AnyOf’ property performs as in (2).

$$\bigcap_{i=1}^K m_i = m_1 \wedge m_2 \wedge \dots \wedge m_K = \begin{cases} T, & \text{if } \forall i \in [1, K], m_i = T \\ F, & \text{if } \exists i \in [1, K], m_i = F \\ IN, & \text{error} \end{cases} \quad (1)$$

$$\bigcup_{i=1}^K m_i = m_1 \vee m_2 \vee \dots \vee m_K = \begin{cases} T, & \text{if } \exists i \in [1, K], m_i = T \\ F, & \text{if } \forall i \in [1, K], m_i = F \\ IN, & \text{error} \end{cases} \quad (2)$$

2) *Rule decision*. Regardless of obligation property, a rule can be abbreviated as $rule(t, e, c)$, where t, e, c are ‘target’, ‘effect’ and ‘condition’. The effect domain of a rule is $E = \{ 'permit', 'deny' \}$. The condition is a list of constraints that request must satisfy, and it shares the same matching result domain V_{MR} with target. We denote the rule decision domain as $V_D = \{ 'P', 'D', 'N', 'IN' \}$, corresponding to ‘Permit’, ‘Deny’, ‘Not-applicable’ and ‘Indeterminate’ respectively. Thus, the mapping of rule decision can be represented as $rule(t, e, c) : V_{MR} \times E \times V_{MR} \rightarrow V_D$, and the decision is illustrated in (3).

$$rule(t, e, c) = \begin{cases} P & \text{if } t \wedge c = T \text{ and } e = 'permit' \\ D & \text{if } t \wedge c = T \text{ and } e = 'deny' \\ N & \text{if } t \wedge c = F \\ IN & \text{error} \end{cases} \quad (3)$$

3) *Policy/Policy set decision*. Denoting a policy as $policy(t, CA(\{rule\}^+))$ and a policy set as $policyS(t, CA(\{policy | policyS\}^+))$. The combining algorithms is a set $CA = \{ 'PO', 'DO', 'FA', 'OOA' \}$ that operates on decision domain V_D , it combines all the decisions made by children rules and policies into one final decision. Formally, let S be a set, $CA(S) : \{V_D\}^{|S|} \rightarrow V_D$. Therefore, the policy and policy set are similar and can be formalized as $policy(t, CA(S)) : V_{MR} \times V_D \rightarrow V_D$. According to different combining algorithms, the policy and policy set decision are concluded in (4).

$$policy(t, CA(S)) = \begin{cases} CA(S) & \text{if } t = T \\ N & \text{if } t = F \text{ or } t = IN \text{ and } CA(S) = N \\ IN & \text{if } t = IN \end{cases} \quad (4)$$

C. A sample XACML

Fig. 1 illustrates a simple XACML policy example P_1 , containing three rules r_1, r_2, r_3 . In this figure, we use brief XML syntax to describe the policy rather than standard XACML format. The resource property in rules reflects to physical data, and the algorithm is mainly constructed on the resource dimension. In this policy, four resources RS1, RS2, RS3 and RS4 are considered, and they have intersections on which rules may conflict and have redundancies.

```

<policy policyID="P1" CA="Deny-Overrides">
  <target>
    <Actions>Read, Write</Actions>
  </target>
  <Rule RuleID="r1" Effect="Permit">
    <target>
      <Subjects>Alice, Bob</Subjects>
      <Resources>RS1, RS2</Resources>
      <Actions>Read, Write</Actions>
    </target>
    <Condition> 8:00<= Time <=12:00 </Condition>
  </Rule>
  <Rule RuleID=" r2" Effect="Permit">
    <target>
      <Subjects>Bob</Subjects>
      <Resources>RS4</Resources>
      <Actions>Read</Actions>
    </target>
  </Rule>
  <Rule RuleID=" r3" Effect="Deny">
    <target>
      <Subjects> Bob, Jim</Subjects>
      <Resources>RS2, RS3</Resources>
      <Actions>Write</Actions>
    </target>
    <Condition>9:00<= Time <=15:00 </Condition>
  </Rule>
</policy>

```

Figure 1. A simple XACML example.

We can formalize the rules into Boolean expressions, for example, r_1 is illustrated in (5).

$$\begin{aligned}
\text{BoolExpression}_{r_1} = & (\text{Subject} = \text{'Alice'} \vee \text{'Bob'}) \\
& \wedge (\text{Resource} = \text{'RS1'} \vee \text{'RS2'}) \\
& \wedge (\text{Action} = \text{'Read'} \vee \text{'Change'}) \\
& \wedge (\text{AnyCondition})
\end{aligned} \tag{5}$$

Assuming a user Bob makes a request for writing to RS2 at 10:00 am. Formally, $\text{req}(\text{Subject} = \text{'Bob'}, \text{Resource} = \text{'RS2'}, \text{Action} = \text{'Write'}, \text{Condition} = \text{'10:00 am.})$. On execution, the target of P_1 matches the request and returns T . Then, r_1 checks its target and conditions, and gives out the ‘permit’ decision as defined in *effect*. However, it continues to match the next rules because the CA of parent policy P_1 is ‘Deny-override’. Accordingly, r_2 does not match the request (outputs ‘N’ as not-applicable) while r_3 returns ‘deny’ as its rule decision. Finally, the policy combines the three results and gives the ‘deny’ decision according to its CA.

III. FINE-GRAINED POLICY OPTIMIZATION ALGORITHM

In this section, we introduce a data fragment algorithm for resource isolation and policy refinement. We build up a three-layer structure of resources, and map the effective policies to each leaf data node so that fine-grained access control is achieved.

A. Data fragmentation

Firstly, we give the definition of *Disjoint set*, based on which we execute the policy projection algorithm.

Definition 1 (Disjoint set) Let $S\{s_1, s_2, \dots, s_n\}$ be a resource set. If $\{\neg \exists res : res \in s_i, res \in s_j\}$ and any operation on s_i will not affect $s_j (i \neq j)$, S is a disjoint set.

Taking the XACML policy in Fig. 1 as example, the four resources (RS1, RS2, RS3, RS4) intersect with each other as shown in Fig. 2. In order to obtain a disjoint resource set $RS(s1, s2, s3, s4, s5, s6)$, we introduce the data fragment algorithm in Algorithm 1.

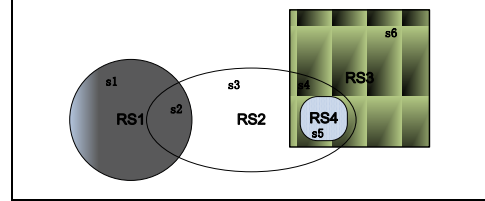


Figure 2. Relationship of resources.

Algorithm 1: Data fragment algorithm

```

// INPUT: a policy.
// OUTPUT: a disjoint set.
Project(policy)
  for each res ∈ GetResources(policy) do
    for each s ∈ RS do
      if res ⊂ s then
        RS.add(s\res);
        RS.replace(s, res); break;
      else if res ⊃ s then
        RS.replace(res, res\s); break;
      else if res ∩ s ≠ ∅ then
        RS.add(s\res);
        RS.replace(s, res ∩ s);
        RS.replace(s, s\res); break;
    RS.add(res);
  Return the resource segment set RS;

```

B. Policy refinement

We build up a three-layer resource tree, in which the physical layer contains all the segments in a disjoint set, while the original policy effects on the logical layer. As shown in Fig. 3, RS1, RS2, RS3 and RS4 relate to (r_1) , (r_1, r_3) , (r_3) and (r_2) , respectively. Based on this structure, we can assign rules to each resource segment in consistency with the original policy and refine policy on segment level.

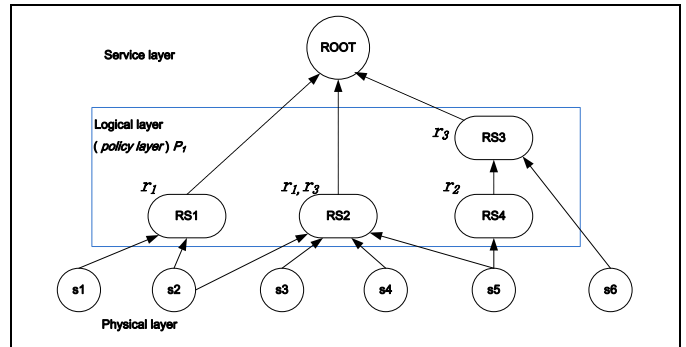


Figure 3. Resource tree.

We define the notation of *rule overlap* (denoting ‘target matching’ as ‘ \models ’) and several RULEs used in the procedure of policy refinement.

Definition 2 (Rule overlap). On a single resource segment, let r_i, r_j be two parallel rules. If $\{\exists req : req \models r_i, req \models r_j\}$, then r_i overlaps with r_j (denoted by $r_i \propto r_j$), and the overlapped part is a rule pair Λ_{r_i, r_j} , consisting of Δ_{r_i} and Δ_{r_j} . Further, if $r_i \propto r_j$ and $r_i.effect = r_j.effect$, then $\Delta_{r_i} = \Delta_{r_j}$.

If deleting Δ_{r_i} (Δ_{r_j}) does not affect the final decision, then Δ_{r_i} (Δ_{r_j}) is removable in this policy. The following RULEs expound principles of removing redundant rules under different combining algorithms.

RULE 1 (CA = Permit-Override)

If $r_i \propto r_j$ and $r_i.effect = permit$, then Δ_{r_j} is removable.

If $r_i \propto r_j$ and $r_i.effect = deny$, then Δ_{r_i} is removable.

RULE 2 (CA = Deny-Override)

If $r_i \propto r_j$ and $r_i.effect = deny$, then Δ_{r_j} is removable.

If $r_i \propto r_j$ and $r_i.effect = permit$, then Δ_{r_i} is removable.

RULE 3 (CA = First-Applicable)

Assuming the sequence of r in policy is $seq(r)$.

If $r_i \propto r_j$ and $seq(r_i) < seq(r_j)$, then Δ_{r_j} is removable.

If $r_i \propto r_j$ and $seq(r_i) > seq(r_j)$, then Δ_{r_i} is removable.

RULE 4 (CA = Only-One-Applicable)

If request on Λ_{r_i, r_j} , the decision is ‘Not Applicable’.

If $r_i \propto r_j$, remove Δ_{r_i} and Δ_{r_j} .

The proofs of above RULEs are similar, and we takes RULE 1 for example, as shown in Fig. 4.

Proof

$\because r_i \propto r_j \therefore \exists \Lambda_{r_i, r_j} : \Delta_{r_i} \in r_i, \Delta_{r_j} \in r_j.$

If $r_i.effect = permit$, while the CA is Permit-Overrides, Δ_{r_j} will be shielded by Δ_{r_i} , no matter $r_j.effect$ is either ‘permit’ or ‘deny’. Thus, removing Δ_{r_j} will not affect the decision, and Δ_{r_j} is removable.

If $r_i.effect = deny$, and if $r_j.effect = deny$, according to the Definition 2, we have $\Delta_{r_i} = \Delta_{r_j}$, remove any one of them is fine. However, if $r_j.effect = permit$, and CA is Permit-Overrides, so Δ_{r_i} will be shielded by Δ_{r_j} . Thus, in both conditions, removing Δ_{r_i} will not affect the decision, and Δ_{r_i} is removable.

Figure 4. Proof of RULE 1.

According to these RULEs, we propose the policy refinement algorithm, as illustrated in Algorithm 2. During the refining procedure, policy is projected to the physical layer, and rules might be refined, removed or kept still.

Algorithm 2: Policy refinement

```

// INPUT: a set of segments bound with rules, the policy CA.
// OUTPUT: refined set of resource segment.
Refine(G, CA)
  for each rule ∈ GetRule(policy) do //bind rules to segment
    for each s ∈ S do
      if s ⊆ GetRelatedResource(rule) then
        bind(rule, s);
  G ← S with bound rules on each element;
  for each g ∈ G do //refine rules on each segment
    for each pair(ri, rj) ∈ C2ruleSet do
      if ri ∝ rj then
        coupleSet.add(Λri, rj); //overlap of ri, rj
    for each Λri, rj ∈ coupleSet do
      case CA = Permit-override then
        Execute by RULE 1;
      case CA = Deny-override then
        Execute by RULE 2;
      case CA = First-applicable then
        Execute by RULE 3;
      case CA = Only-one-applicable then
        Execute by RULE 4;
  Return the refined set G;

```

As for the situation of refining a policy set, the algorithm could be specified recursively for each children policy.

C. Algorithm performance analysis

We analyze the fine-grained policy optimization algorithm on computation overhead and storage overhead. Basically, we suppose that a policy P contains K rules, N resources and M resource segments, which are generated by the data fragment algorithm, and on each segment i , there exist H_i rules and C_i conflicts.

1) Computation overhead

In the data fragment algorithm, it costs $O(N \log_2 M)$ to obtain resource segmentation set, where M varies upon the coupling degree among resources.

Nonetheless, in the procedure of policy refinement, the cost of policy projection is $O(KM)$, which is decided by the number of segments $\sum_{i \in M} s_i$ and related rules $\sum_{k \in K} r_k$. The overhead of refining an individual segment relies on finding rule conflict pairs, which contributes to $O(H_i \log_2 H_i)$, while resolving rule conflicts takes $O(L_i)$. Thus, the complexity of refining a resource segment set is the accumulation of cost on each element, resulting in $O(\sum_{i \in M} H_i \log_2 H_i + L_i)$.

Finally, the computation overhead of our approach is $O(N \log_2 M + KM + \sum_{i \in M} (H_i \log_2 H_i + L_i))$.

2) Storage overhead

We define the function $W(R)$ to measure the physical storage size. When we execute the algorithm, the overhead of storage is $\Theta(\sum_{i \in M} W(R_i) + \sum_{i \in M} H_i \cdot W(rule_i))$.

Our approach advances in storage compared with original policies. In original policy, each resource is considered as a single entity, and the total size of all resources is $\sum_{i=1}^N W(R_i)$. However, by developing the relationship among resources, we extract the common parts of resources. As a result, the storage size is reduced by $\sum_{i=1}^N W(R_i) - \sum_{i=1}^M W(S_i)$.

IV. A CASE STUDY

The policy based access control methods can be applied in many fields such as banking, healthcare, ATM and market etc. to achieve data security and user privacy [17] [18] [19]. We apply the fine-grained policy optimization algorithm to data access control in cloud computing, and Fig. 5 describes the framework of a healthcare records management system.

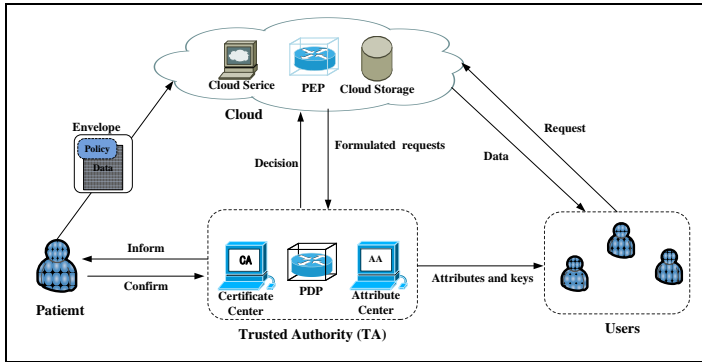


Figure 5. Application scenario on healthcare system.

Assume that Alice is a patient, and she has well processed her private healthcare record with policies before outsourcing to the cloud. Bob is medical researcher in university, and he needs patient's health records for study. Once Bob requests to

cloud for the statistics with his identity and public attributes, the cloud transforms the request into expressive XML format and sends it to the TA. Then, the PDP in TA makes the decision according to the defined policies and inform Alice of the result. Upon receiving Alice's acknowledgement, TA sends the decryption key to Bob through secure channels. Thus, Bob can have access to Alice private data under the conditions defined in the policies.

To evaluate the performance of our proposal, we utilize the policies with different amount of rules, different coupling degree of rules and resources. The coupling of rules leads to a certain number of conflicts, and the coupling of resources decides the number of resource segments. Our experiments were founded on the API of SUN-XACML and performed on Intel(R) Core(TM) i3-2330M CP U 2.30 GHz with 2.67-GB RAM running on Win7.

We generate synthetic policies for most situations and compare the decision efficiency with the existing methods, *Simple PDP* [3] and *Melcoe PDP* [20]. The *Simple PDP* adopts a list structure to traverse rules for matching, and *Melcoe PDP* employs category of data attributes. The statistics of two representative situations are listed in Table I and Table II. We use a triple $\langle \text{Few/Many, Few/Many, Few/Many} \rangle$ to simply express the amount of *rules*, *conflicts* and *segments*. Fig. 6 illustrates the experiment results of all the situations.

TABLE I. DECISION EFFICIENCY UNDER $\langle \text{FEW, FEW, FEW} \rangle$

Policy parameters				PDPs evaluation (ms)		
Policy#	Rule#	Res#	Seg#	Simple PDP	Melcoe PDP	Our PDP
10	20	60	62	33.8608	26.7024	63.3365
20	40	60	62	51.6174	41.0145	65.0015
30	60	60	62	67.3090	54.8726	69.1087
40	80	60	62	82.0385	69.2740	75.7588
50	100	60	62	98.6908	81.0534	83.5972

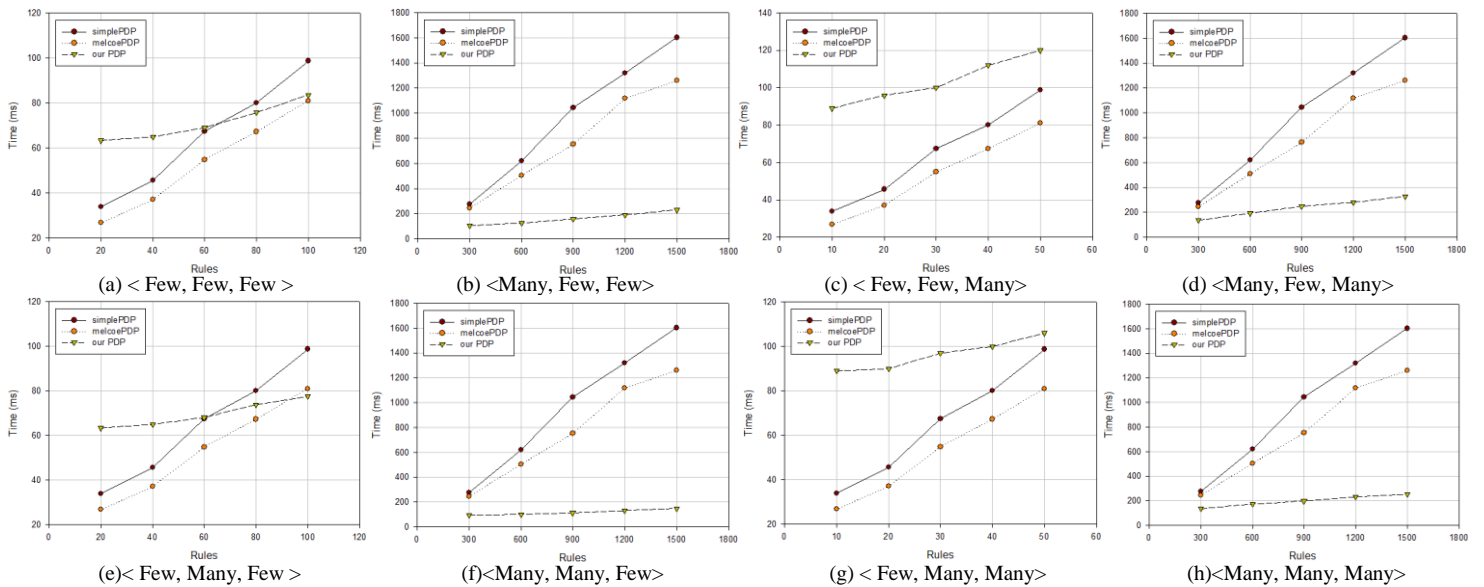


Figure 6. Efficiency Impacts on amount of rules, conflicts and resource intersections.

TABLE II. DECISION EFFICIENCY UNDER <MANY, MANY, MANY >

Policy parameters				PDPs evaluation (ms)		
Policy#	Rule#	Res#	Seg#	Simple PDP	Melcoe PDP	Our PDP
10	300	60	155	276.5745	244.7562	135.6794
20	600	60	155	620.4870	509.0062	172.6407
30	900	60	155	1043.7546	761.0980	197.6577
40	1200	60	155	1319.8039	1117.6535	231.5092
50	1500	60	155	1602.8325	1259.0271	253.2671

Through the experiments, we conclude that the decision efficiencies of *Simple PDP* and *Melcoe PDP* depend on the amount of rules, with little concerning about the coupling degree of rules and resources. In the contrast, our approach has great advantages in the situation of large amount of rules. We also exceed traditional methods in multi-resource requests, since the redundancies of data are eliminated in the segmentation phase.

V. CONCLUSION

We have proposed an innovative mechanism of facilitating data security for cloud resource service. The fine-grained policy optimization algorithm projects the policy to the resource dimension, and refines rule on each individual resource segment. We can encrypt the sensitive data and attach sticky policy to ensure that the data is processed or handled according to customers' willing.

The fragmentation of resource decomposes data into obfuscated segments to protect the physical entities, while available services are provided in service layer and logical layer. Cloud users may request resources by names, without knowing the components or physical locations of the resources, and they can only get those identity-permitted data. We have discussed the performance of our proposal, in terms of the amount of rules, conflicts and segments. Through the experiment, we conclude that our approach has great advantages in large scale of policies.

We would develop a prototype and explore how our strategy can be applied to other fields concerning about access control and security.

ACKNOWLEDGMENT

This work was partially supported by the NSF of China under grants No. 61173048 and No. 61300041, Specialized Research Fund for the Doctoral Program of Higher Education under grant No. 20130074110015, and the Fundamental Research Funds for the Central Universities under Grant No.WH1314038.

REFERENCES

- [1] W. Jansen and T. Grance, "Guidelines on Security and Privacy in Public Cloud Computing", NIST Special Publication, pp. 800-144, 2011.
- [2] H. Takabi, J. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments", *IEEE Security and Privacy*, vol. 6, no. 6, pp. 24-31, 2010.
- [3] S. Godik and T. Moses, "eXtensible Access Control Markup Language (XACML) Version 1.1", OASIS, 2003.
- [4] M. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: sticky policies and enforceable tracing services", *Database and Expert Systems Applications (DESA)*, pp. 377-382, 2003.
- [5] S. Pearson and M. Mont, "Sticky policies: an approach for managing privacy across multiple parties", *IEEE Computer*, vol. 44, no. 9, pp. 60-68, 2011.
- [6] S. Trabelsi and J. Sendor, "Sticky policies for data control in the cloud", *IEEE PST*, pp. 75-80, 2012.
- [7] G. Ahn, H. Hu, J. Lee, and Y. Meng, "Representing and reasoning about web access control policies", *IEEE Software and Applications*, pp. 137-146, 2010.
- [8] H. Hu and G. Ahn, "Discovery and resolution of anomalies in web access control policies", *IEEE Dependable and Secure Computing*, vol. 10, no. 6, pp. 341-354, 2013.
- [9] Y. Wang, D. Feng, and L. Zhang, "XACML policy evaluation engine based on multi-level optimization technology", *Journal of Software*, vol. 22, no. 2, pp. 323-338, 2011.
- [10] M. Said, M. Shehab, and S. Anna, "Adaptive reordering and clustering-based framework for efficient XACML policy evaluation", *IEEE Service Computing*, vol. 4, no. 4, pp. 300-313, 2011.
- [11] D. Lin, P. Rao, R. Ferrini, and E. Bertino, "A similarity measure for comparing XACML policies", *IEEE Knowledge and Data Engineering*, vol. 25, no. 9, pp. 1946-1959, 2013.
- [12] A. Bertolino, S. Daoudagh, and F. Ionetti, "Automated testing of eXtensible Access Control Markup Language-based access control systems", *IET Software*, vol. 7, no. 4, pp. 203-212, 2013.
- [13] D. Agrawal, J. Giles, K. W. Lee, and J. Lobo, "Policy ratification", *IEEE Policies for Distributed Systems and Networks*, pp. 223-232, 2005.
- [14] X. Wu and P. Qian, "A verification for PDAC model by policy language", *ICCSE*, pp. 14-17, 2012.
- [15] G. Bruns, D. Dantas, and M. Huth, "A simple and expressive semantic framework for policy composition in access control", *Formal methods in Security Engineering*, *ACM*, pp. 12-21, 2007.
- [16] C. Ngo, Y. Demchenko, and C. D. Laat, "Decision Diagrams for XACML Policy Evaluation and Management", *Computers & Security*, vol. 49, no. 1, pp. 1-16, 2015.
- [17] M. Ghorbel, A. Aghasaryan, and M. P. Dupont, "A multi-environment application of privacy data envelopes", *Policies for Distributed Systems and Networks*, pp. 180-181, 2011.
- [18] M. Li, S. Yu, and Y. Chen, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption", *IEEE Parallel and Distributed Systems*, vol. 24, no. 1, pp. 131-143, 2013.
- [19] Asela, "Banking sample with XACML", <http://xacmlinfo.org/2014/03/11/atm-banking-sample-with-xacml/>, 2014.
- [20] Jajodia and P. Samarath, "A logical language for expressing authorizations", *IEEE Security and Privacy*, pp. 31-42, 1997.

A Reliable and Secure Cloud Storage Schema Using Multiple Service Providers

Haiping Xu and Deepti Bhalerao

Computer and Information Science Department
University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA
{hxu, dbhalerao}@umassd.edu

Abstract—Despite the many advantages provided by cloud-based storage services, there are still major concerns such as security, reliability and confidentiality of data stored in the cloud. In this paper, we propose a reliable and secure cloud storage schema using multiple service providers. Different from existing approaches to achieving data reliability using redundancy at the server side, we propose a reliable and secure cloud storage schema that can be implemented at the client side. In our approach, we view multiple cloud-based storage services as virtual independent disks for storing redundant data encoded using erasure codes. Since each independent cloud service provider has no access to a user’s complete data, the data stored in the cloud would not be easily compromised. Furthermore, the failure or disconnection of a service provider will not result in the loss of a user’s data as the missing data pieces can be readily recovered. To demonstrate the feasibility of our approach, we developed a prototype cloud-based storage system that breaks a data file into multiple data pieces, generates an optimal number of checksum pieces, and uploads them into multiple cloud storages. Upon the failure of a cloud storage service, the application can quickly restore the original data file from the available pieces of data. The experimental results show that our approach is not only secure and fault-tolerant, but also very efficient due to concurrent data processing.

Keywords—Cloud storage; reliability; data security; erasure codes; cloud service provider; integer linear programming.

I. INTRODUCTION

As an ever-growing data storage solution, cloud-based storage services have become a highly practical way for both people and businesses to store their data online. The pay-as-per-use model of cloud computing eliminates the upfront commitment from cloud users; thereby it allows users to start small businesses quickly, and increase resources only when they are needed. However, since data storage locations and security measures at the server site are typically unknown, most of the users have not yet become comfortable with exploiting the full potential of the cloud. Many incidents happened recently have made users question the reliability of cloud storage services. For example, in May 2014, Adobe’s ID service went down, leaving Creative Cloud users locked out of their software and account for over 24 hours [1]. In early 2013, Dropbox service had a major cloud outage that kept users offline and unable to synchronize using their desktop apps for more than 15 hours [2]. Prolonged cloud data service outages and security concerns can be fatal for businesses with

data critical domains such as healthcare, banking and finance. Today, almost all the cloud service providers (CSP) have implemented fault-tolerant mechanisms at their server sides to recover original data from service failure or data corruption. Such mechanisms are suitable at the time of scheduled maintenance or for a small number of hard disk failures. However, they are of no use for the end users to ensure the reliability and security of their cloud data when major cloud services fail or the cloud services have been compromised. Hence, to achieve high reliability and security of critical data, users should not depend upon a single cloud service provider. In this paper, we propose an approach that can provide security and fault tolerance to the user’s data from the client side. In our approach, we decompose an original data file into multiple data pieces, and generate checksum pieces using erasure codes [3]. The pieces of data are spread across multiple cloud services, which can be retrieved and combined to recover the original file. We achieve data redundancy in our approach using erasure codes at the software level across multiple cloud service providers. Therefore, the original data can be recovered even when there is a cloud outage where some cloud service fails completely. Using this approach, user’s data would not be easily compromised by unauthorized access and security breach, as no single cloud service has the complete knowledge of user’s data. Thus, users could have the sole control of their cloud data, and do not need to rely on the security measures provided by cloud service providers. Finally, to improve the network performance of our approach, we adopt the multithreading technology, and fully utilize the network bandwidth in order to minimize the time required to access data over the cloud.

There have been many research efforts on using erasure codes at the server side to make cloud storage service reliable. Huang *et al.* proposed to use erasure codes in Windows Azure storage [4]. They introduced a new set of codes for erasure codes called Local Reconstruction Codes (LRC) that could reduce the number of erasure coding fragments required for data reconstruction. Gomez *et al.* introduced a novel persistency technique that leverages erasure codes to save data in a reliable fashion in Infrastructure as a Service (IaaS) clouds [5]. They presented a scalable erasure coding algorithm that could support a high degree of reliability for local storage with the cost of low computational overhead and a minimal amount of communication. Khan *et al.* provided guidance for deploying erasure coding in cloud file systems to support load balance and incremental scalability in data centers [6]. Their proposed approach can prevent correlated failures with data

loss and mitigate the effect of any single failure on a data set or an application. Although the above approaches can significantly enhance the reliability of cloud data at data centers, they provide no support for end users to deal with failures or cloud outage of the service providers. Different from the existing approaches, we apply erasure-coding techniques at the application level using multiple cloud service providers. By deploying user's encoded redundant data across multiple cloud storage services, our approach is fault tolerant for cloud storage when any of the cloud services fails.

There is also a considerable amount of work on securing cloud data, to which this work is closely related. Santos *et al.* proposed a secure and trusted cloud computing platform (TCCP) for IaaS providers such as Amazon EC2 [7]. The platform provides a closed box execution environment that guarantees confidential execution of guest virtual machines on a cloud infrastructure. Hwang and Li proposed to use data coloring and software watermarking techniques to protect shared cloud data objects [8]. Their approach can effectively prevent data objects from being damaged, stolen, altered, or deleted, and users may have their sole access to their desired cloud data. The existing approaches to securing cloud data typically assume that the cloud service providers are trustable and they can prevent physical attacks to their servers. However, this might not be true in reality, as service providers typically tend to collect users' cloud data for their commercial purposes such as targeted advertising. Furthermore, there have been many incidents that cloud service providers were compromised by either internal or external hackers, and thousands of users' critical data were compromised. Therefore, merely relying on service providers' security mechanisms is not a feasible solution for both people and businesses to store their critical data in the cloud. It is required that users should be allowed to apply security mechanisms to their own data at the client side. Different from the aforementioned methods to securing cloud data at the server side, our approach does not rely on any security measures supported by the service providers. Instead, the cloud storage application running at the client side can split users' data into pieces, encode them using erasure codes, and distribute them to multiple service providers. As no single CSP has its access to a user's entire data, user's data are much securer than those stored with a single cloud service.

In this paper, we extend the methodology and results of a preliminary study on secure and fault-tolerant model of cloud information storage [9]. In the previous work, we followed the RAID (Redundant Array of Independent Disks) approach to encode user's data using XOR parity, and developed a hierarchical colored Petri nets (HCPN) model for secure and fault-tolerant cloud information storage systems. In this paper, we adopted erasure codes to achieve fault tolerance for cloud data, and presented a detailed design for a reliable and secure cloud storage schema. To demonstrate the effectiveness of our proposed approach, we implemented a prototype using three major cloud service providers (i.e., Amazon, Google and Dropbox), which allows users to securely, reliably and efficiently store their critical data in the cloud.

II. RELIABLE AND SECURE CLOUD DATA STORAGE

To address the aforementioned major issues in cloud storage services, we propose a reliable and secure cloud storage

schema using multiple CSPs. Figure 1 shows a framework for such a system. The major component of the system is the cloud storage application that uses erasure codes to encode and decode file pieces at the client side, and upload and download encoded file pieces concurrently at multiple cloud services. As shown in the figure, when a user wants to upload a file into the cloud, the application first splits the file into multiple data pieces, say n pieces, and then encode them into an optimal number of m checksum pieces using the erasure coding technique. Once the data pieces and checksum pieces are ready, they are concurrently uploaded into multiple cloud storages maintained by different CSPs, noted as CSP_1 , CSP_2 , ..., and CSP_N in Fig. 1. As none of the CSPs has the complete knowledge about the user data, this approach can effectively defend against data breach from any single CSP.

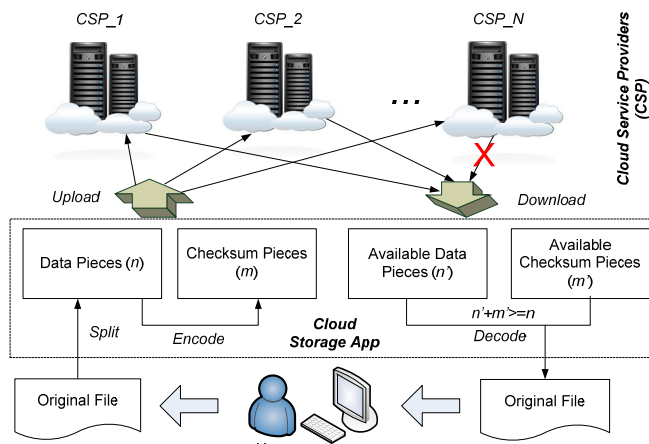


Figure 1. A framework for reliable and secure cloud storage systems

On the other hand, when a user wants to download a stored file, the application will first try to download the n data pieces from the multiple cloud storages concurrently. If all data pieces are available, they can be efficiently combined into the original file without any additional decoding process. However, in the case when one or more service provider fails, the application must automatically download all available data pieces (n') and available checksum pieces (m'). As long as $n' + m' \geq n$, due to the erasure coding technique, the application can always successfully decode the missing data pieces using the available pieces of data, and restore the original file. Note that the checksum pieces serve as the redundancy of the original file, which makes our approach reliable and fault tolerant.

III. ERASURE CODES AND REED-SOLOMON CODING

A. Erasure Codes

In early days, fault tolerance of cloud data is commonly achieved through simple data replication. Multiple copies of original data have to be maintained on different cloud servers in order to make data more reliable. However, data replication now becomes highly unfeasible due to its low space efficiency and the ever-increasing amount of cloud data. Erasure codes, also known as forward error correction (FEC) codes, manage to overcome the disadvantages of the data replication approach, and can achieve a high degree of fault tolerance with a much lower cost of physical storage [3]. An erasure code takes n data

words and transforms them into m code words such that any n out of $(n+m)$ words are enough to recover the original n data words. Erasure codes use a mathematical function to convert original data words into encoded words and to recover them back. They can be very efficient in providing fault tolerance for large quantities of data, hence they are quite suitable for large-scale cloud storage systems.

Data redundancy through parity codes represents the simplest form of erasure codes, which overcomes the drawback of data replication. RAID-5 is the most commonly used technique that uses parity codes. It calculates parities from the original data to achieve fault tolerance. However, this technique is typically used by CSPs at the hardware level, and very few research efforts attempted to apply the RAID concept at the software level to resolve issues related to the major data failures of a service provider, which actually have become quite common nowadays [9].

B. Reed-Solomon Coding for Cloud Based Storage

Use of error-correction codes for redundancy has become prevalent due to its various advantages. Reed-Solomon (RS) coding is a type of optimal erasure codes, which follows the basic error-correction techniques. There are many different ways to implement error-correction using erasure codes, but RS technique is a good compromise between efficiency and complexity [10]. Traditionally, RS technique has been used in various applications such as error-correction in CD-ROM and DVDs, satellite communications, digital television, and wireless or mobile communications [11]. The use of RS technique to provide fault tolerance over the cloud is a fairly new idea. Our approach to distributing data and checksum pieces with multiple cloud services could build a RAID-like system with less storage overhead and more flexibility in the degree of fault tolerance for the stored data. Here we first briefly introduce the RS coding approach. Let there be n data pieces, we encode all data pieces using RS algorithm into m checksum pieces such that out of $(n+m)$ pieces, any n pieces are enough to recover the original n data pieces. If the $(n+m)$ pieces of data are distributed over $(n+m)$ devices, this algorithm can be used to handle m failures of the devices.

To simplify matters, we assume each data piece is an unsigned byte ranged from 0 to 255. In order to calculate the checksum bytes, we first create an $(m+n) \times n$ Vandermonde matrix A , where the i, j -th element of A is defined to be i^j [11]. By this definition, when m rows are deleted from A , the newly formed matrix is invertible. Then we derive the information dispersal matrix B from A using a sequence of elementary matrix transformation. The information dispersal matrix B is defined as in Eq. (1), where I is an $n \times n$ identity matrix, and F is an $m \times n$ matrix. Note that since elementary matrix transformation does not change the rank of a matrix and each row in A is linearly independent, the information dispersal matrix B maintains the property that when m rows are deleted from B , the newly formed matrix is invertible.

$$B = \begin{bmatrix} I \\ F \end{bmatrix} \quad (1) \quad BD = \begin{bmatrix} I \\ F \end{bmatrix} D = E, \text{ where } E = \begin{bmatrix} D \\ C \end{bmatrix} \quad (2)$$

Let D be a vector of n -byte data d_0, d_1, \dots, d_{n-1} , and C be a vector of m -byte checksum c_0, c_1, \dots, c_{m-1} . With the information dispersal matrix B , we can calculate the checksum

vector C from the data vector D as in Eqs. (3), where $f_{i,j}$ ($0 \leq i \leq m-1, 0 \leq j \leq n-1$) are elements of the $m \times n$ matrix F . Based on the calculation of C , Eq. (2) must hold.

$$\begin{aligned} c_0 &= f_{0,0} * d_0 + f_{0,1} * d_1 + \dots + f_{0,n-1} * d_{n-1} \\ c_1 &= f_{1,0} * d_0 + f_{1,1} * d_1 + \dots + f_{1,n-1} * d_{n-1} \\ &\dots \\ c_{m-1} &= f_{m-1,0} * d_0 + f_{m-1,1} * d_1 + \dots + f_{m-1,n-1} * d_{n-1} \end{aligned} \quad (3)$$

Now suppose k bytes, where $k \leq m$, are missing from vector D . By deleting the missing k elements from D as well as any $m-k$ elements from C , we derive a new n -byte vector E' as in Eq. (4), where D' is a $(n-k)$ -byte vector $d'_0, d'_1, \dots, d'_{n-k-1}$, and C' is a k -byte vector $c'_0, c'_1, \dots, c'_{k-1}$. Similarly, in Eq. (2), by deleting m rows from B that correspond to the deleted rows in E , we derive an $n \times n$ matrix B' as defined in Eq. (5), where I' is an $(n-k) \times n$ matrix, and F' is a $k \times n$ matrix. The matrix B' must be invertible as we have mentioned, and Eq. (6) must hold.

$$E' = \begin{bmatrix} D' \\ C' \end{bmatrix} \quad (4) \quad B' = \begin{bmatrix} I' \\ F' \end{bmatrix} \quad (5) \quad B'D = \begin{bmatrix} I' \\ F' \end{bmatrix} D = \begin{bmatrix} D' \\ C' \end{bmatrix} \quad (6)$$

By calculating the inverse matrix $G = B'^{-1}$ using Gaussian elimination method, we can recover the data vector D as in Eqs. (7), where $g_{i,j}$ ($0 \leq i \leq n-1, 0 \leq j \leq n-1$) are elements of the $n \times n$ matrix G .

$$\begin{aligned} d_0 &= g_{0,0} * d'_0 + g_{0,1} * d'_1 + \dots + g_{0,n-k-1} * d'_{n-k-1} + \\ &\quad g_{0,n-k} * c'_0 + g_{0,n-k+1} * c'_1 + \dots + g_{0,n-1} * c'_{k-1} \\ d_1 &= g_{1,0} * d'_0 + g_{1,1} * d'_1 + \dots + g_{1,n-k-1} * d'_{n-k-1} + \\ &\quad g_{1,n-k} * c'_0 + g_{1,n-k+1} * c'_1 + \dots + g_{1,n-1} * c'_{k-1} \\ &\dots \\ d_n &= g_{n-1,0} * d'_0 + g_{n-1,1} * d'_1 + \dots + g_{n-1,n-k-1} * d'_{n-k-1} + \\ &\quad g_{n-1,n-k} * c'_0 + g_{n-1,n-k+1} * c'_1 + \dots + g_{n-1,n-1} * c'_{k-1} \end{aligned} \quad (7)$$

Once the n -byte vector D is restored, the m -byte vector C can be recalculated as in Eqs. (3). Note that implementation of the RS algorithm for data files requires to perform computations on binary words of a fixed length w . For example, when the binary word is a byte, w equals 8. To ensure that the RS algorithm works correctly for fixed-size words, all arithmetic operations must be performed over Galois Fields with 2^w elements denoted as $GF(2^w)$ [11]. A Galois field $GF(2^w)$ is also known as a finite field which contains finitely many elements, namely 0, 1, ..., 2^w-1 . Arithmetic operations performed over Galois Fields will result in finite values in $GF(2^w)$. As such, all arithmetic operations mentioned in this section, including the matrix inverse, encoding and recovery of data, must be calculated using Galois Fields arithmetic.

IV. OPTIMAL NUMBER OF CHECKSUM PIECES

A. Calculating the Optimal Number of Checksum Pieces

In order to achieve the highest space efficiency in our approach, we propose a procedure to compute the minimal number of checksum pieces that allows the failures of multiple cloud service providers. Let N be the number of service

providers, $\Gamma = \{1, 2, \dots, N\}$, and M be the maximal number of services allowed to fail or become unavailable at the same time, where $1 \leq M \leq N-1$. We define a failure set Φ as follows:

$$\Phi \in \mathcal{P}(\Gamma), \text{ where } \mathcal{P}(\Gamma) \text{ is the power set of } \Gamma, \text{ and } |\Phi| \leq M.$$

The set of available CSPs Ω corresponding to Φ can be defined as in Eq. (8).

$$\Omega = \Gamma - \Phi \quad (8)$$

Let the number of data pieces of a file be n . In order to distribute n data pieces evenly over N cloud service providers, we calculate the number of data pieces n_1, n_2, \dots , and n_N stored at $CSP1, CSP2, \dots$, and CSP_N , respectively, as in Eq. (9).

$$n_i = \begin{cases} \lceil n/N \rceil & \text{when } i=1 \\ \left\lfloor (n - \sum_{j=1}^{i-1} n_j) / (N - i + 1) \right\rfloor & \text{when } 1 < i < N \\ n - \sum_{j=1}^{N-1} n_j & \text{when } i=N \end{cases} \quad (9)$$

where $n = n_1 + n_2 + \dots + n_N$. Eq. (9) allows even distribution of n data pieces over N cloud service providers such that $|n_i - n_j| \leq 1$ for $1 \leq i, j \leq N$. For example, when $N = 3$ and $n = 7$, the number of data pieces distributed over three cloud service providers $CSP1, CSP2$, and $CSP3$ will be 3, 2, 2, respectively.

As a major requirement for fault tolerance, when up to M CSPs become unavailable, the original data must be recovered from the remaining CSPs from the available set Ω . Let m be the number of checksum pieces required, and m_1, m_2, \dots, m_N are the numbers of checksum pieces distributed over $CSP1, CSP2, \dots$, and CSP_N , respectively. Obviously, we have $m = m_1 + m_2 + \dots + m_N$. To calculate the minimal number of checksum pieces m , we can solve the integer linear programming problem as defined in Eq. (10).

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^N m_i \\ & \text{subject to} && \text{for each failure set } \Phi \\ & && \sum_{i \in \Omega} m_i \geq \sum_{j \in \Phi} n_j \\ & && \text{where } \Phi \in \mathcal{P}(\Gamma) \text{ and } |\Phi| = M \end{aligned} \quad (10)$$

Note that a solution to the above optimal problem automatically satisfies the cases when $|\Phi| < M$. The space efficiency e of a solution can be calculated as in Eq. (11).

$$e = 1 - m/(n + m), \text{ where } n = \sum_{i=1}^N n_i \text{ and } m = \sum_{i=1}^N m_i \quad (11)$$

As an example, let $N = 3$ and $M = 1$, the integer linear programming problem can be simplified as in Eq. (12).

$$\begin{aligned} & \text{minimize} && m_1 + m_2 + m_3 \\ & \text{subject to} && m_1 + m_2 \geq n_3 \quad // \text{ when } \Phi = \{3\} \\ & && m_2 + m_3 \geq n_1 \quad // \text{ when } \Phi = \{1\} \\ & && m_1 + m_3 \geq n_2 \quad // \text{ when } \Phi = \{2\} \end{aligned} \quad (12)$$

Table 1 shows the optimal solutions and their space efficiency for the above example with n ranges from 2 to 14. For instance, when $n = 8$ ($n_1 = 3, n_2 = 3, n_3 = 2$), the optimal solution is $m_1 = 1, m_2 = 1$, and $m_3 = 2$, and the space efficiency $e = 1 - 4/(8+4) = 0.6667$. In this case, if any service provider becomes unavailable, the missing 4 pieces of data can always be recovered from the remaining data and checksum pieces stored at the other two CSPs.

Table 1. Optimal number of checksum pieces and space efficiency

Data Pieces (n)	(n ₁ , n ₂ , n ₃)	(m ₁ , m ₂ , m ₃)	Checksum Pieces (m)	Space Efficiency (e)
2	(1, 1, 0)	(0, 0, 1)	1	0.6667
3	(1, 1, 1)	(0, 0, 1)	2	0.6000
4	(2, 1, 1)	(0, 1, 1)	2	0.6667
5	(2, 2, 1)	(1, 1, 1)	3	0.6250
6	(2, 2, 2)	(1, 1, 1)	3	0.6667
7	(3, 2, 2)	(1, 2, 1)	4	0.6364
8	(3, 3, 2)	(1, 1, 2)	4	0.6667
9	(3, 3, 3)	(2, 2, 1)	5	0.6429
10	(4, 3, 3)	(1, 2, 2)	5	0.6667
11	(4, 4, 3)	(2, 2, 2)	6	0.6471
12	(4, 4, 4)	(2, 2, 2)	6	0.6667
13	(5, 4, 4)	(2, 3, 2)	7	0.6500
14	(5, 5, 4)	(3, 3, 2)	7	0.6667

B. Distribution of Data and Checksum Pieces over CSPs

When dealing with a file with k bytes, if k is not a multiple of n , we first need to append r bytes with random values to the end of the file such that $((k+r) \bmod n) = 0$. Then we group the $(k+r)$ bytes into n data pieces so that each of them contains $(k+r)/n$ bytes. By applying Eq. (9) and Eq. (10), we calculate the distribution of the n data pieces and the optimal number of checksum pieces. Finally, using the equations described in Section III.B, we can calculate the checksum pieces. Figure 2 shows an example of file distribution at service providers $CSP1, CSP2$ and $CSP3$ when $N = 3, M = 1, n = 8$ and $m = 4$.

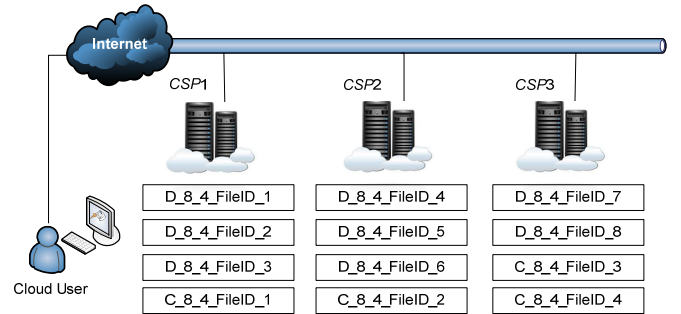


Figure 2. Distribution of data and checksum pieces at three CSPs

As shown in Fig. 2, we distribute 3, 3, 2 data pieces (denoted by the file names starting with the letter “D”) over $CSP1, CSP2$ and $CSP3$, respectively. Based on the optimal solution given in Table 1, we also distribute 1, 1, 2 checksum pieces (denoted by the file names starting with the letter “C”) over $CSP1, CSP2$ and $CSP3$, respectively. When any of the service providers fails, the original data can be recovered from the remaining 8 pieces of data using Eq. (7). It is worth noting that by the definition of the RS coding technique, when up to 4 pieces of data from multiple CSPs are missing or corrupted, the original file can still be recovered using Eq. (7).

V. CASE STUDY

To demonstrate the feasibility of our proposed approach, we developed a prototype secure and reliable cloud storage application in Java. We adopt three different cloud services supported by major CSPs to store our data pieces and checksum pieces in the cloud. The selected cloud services are Amazon S3, Google App Engine, and Core Dropbox APIs

with free user accounts. All experiments have been conducted with excellent Internet connections at University of Massachusetts Dartmouth, where the download speed was around 160 Mbps (~20MB/s) and the upload speed was around 400 Mbps (~50MB/s). Therefore, the network connection at the client side will not become a bottleneck for all of our experiments. As shown in Fig. 3, the user interface of the application allows one to select a file to upload into the cloud. After choosing the number of data pieces (n), the optimal number of checksum pieces (m) can be automatically calculated using integer linear programming. By clicking on the “Encode and Upload” button, the selected file is divided into n data pieces, and the application automatically encodes them into m checksum pieces. Once all pieces of data become ready, they are uploaded into the three selected cloud storage services using multithreading techniques. The message box in the user interface displays the encoding time, the uploading time and the total processing time.

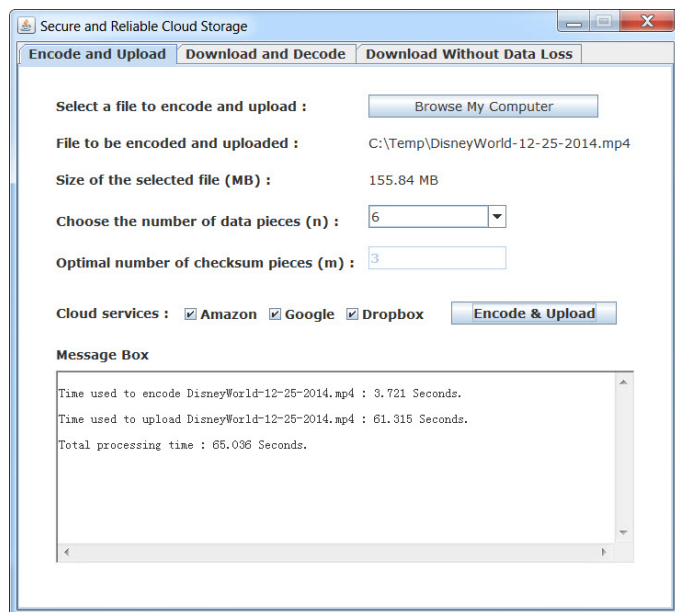


Figure 3. Encode and upload a file to multiple cloud storage services

Figure 4 shows the user interface for downloading and decoding an uploaded file in the cloud. As shown in the figure, a user first selects a file from the list of uploaded files, then chooses at least two cloud service providers as the maximal number of failed cloud services M equals 1. By clicking on the “Download and Decode” button, the available file pieces are concurrently downloaded to the local computer, where the original file is recovered using RS coding techniques. Similarly, as shown in Fig. 4, the message box in the user interface displays the downloading time, the decoding time and the total processing time, as well as the location of the downloaded file on the user’s local computer.

To analyze the performance of our approach, we selected a video file with a file size of 156 MB. Figure 5 shows the encoding and uploading time vs. the number of data pieces set by the user. From the figure, we can see that when we increase the number of data pieces from 2 to 8, the uploading time drops down significantly; while the encoding time has slightly increased. The significant performance improvement for

uploading is due to the use of multithreading techniques; however, the increased number of data pieces along with more checksum pieces result in more overhead for encoding. When the number of data pieces n is further increased, the uploading time dramatically goes up. Based on our further experiments with the cloud service providers, the concurrent processing capabilities of the service providers as well as their bandwidths become a major issue when the number of concurrent uploading reaches 5. Note that when $n = 10$, the optimal number of checksum pieces $m = 5$, so the number of concurrent uploading to each CSP is 5.

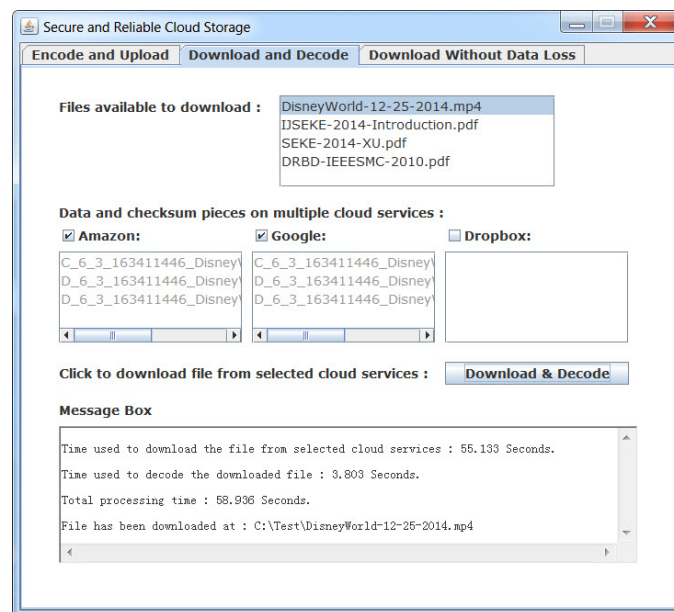


Figure 4. Download and decode a file from clouds with a failed service

Figure 6 shows the downloading and decoding time vs. the number of data pieces set by the user. From the figure, we can see that when we increase the number of data pieces from 2 to 9, the downloading time drops down significantly; while the decoding time has slightly increased. Similar to the case of uploading, the significant performance improvement for downloading is also due to the multithreading techniques, and the increased number of data pieces along with more checksum pieces result in more overhead for decoding. When the number of data pieces n is further increased, the downloading time goes up slightly, which it is not as bad as in the uploading case with dramatic performance change. This is because major cloud service providers typically put more restrictions on their upload bandwidths than their download bandwidths, especially for free user accounts.

From the above experimental results, we can see that both the uploading and downloading time can be significantly reduced by selecting a reasonable number of data pieces. For example, when a file size is between 100 to 200 MB, based on our experiments, the number of data pieces should normally be set to 8 as long as the network bandwidth is sufficient. According to Table 1, when $n = 8$, the optimal number of checksum pieces $m = 4$. In this case, the space efficiency e reaches its highest value 0.6667. It is worth noting that when no service provider fails, the application only needs to download the data pieces, and no checksum pieces are needed

for restoring the original file. In this case, the downloading time can be further reduced, and the decoding time becomes merely the time needed to combine the data pieces into the original file. Therefore, in a normal case with no failure of service providers, the overall performance for file retrieval will be better than the results demonstrated in Fig. 6.

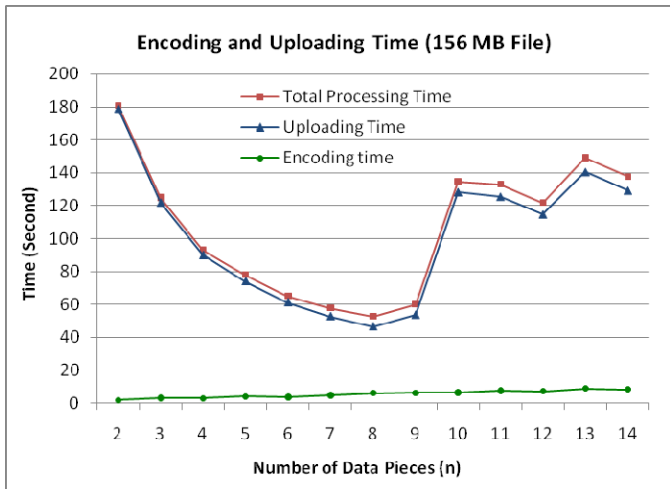


Figure 5. Encoding & uploading time vs. number of data pieces

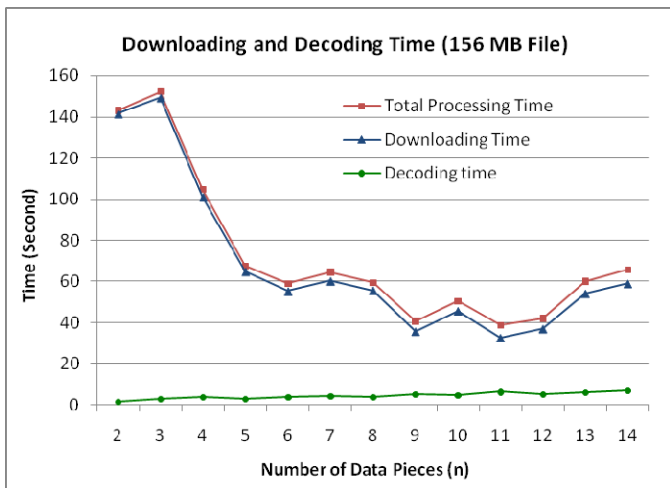


Figure 6. Downloading and decoding time vs. number of data pieces

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed three major issues with cloud storage, namely reliability, security and performance. Instead of achieving data reliability using redundancy at the server side, we presented a reliable and secure cloud storage schema for end users. In our approach, we view multiple cloud storage services as virtual disks, and upload redundant data files into multiple cloud storages. The redundant data files are calculated using erasure codes techniques, which allow multiple failures of the data pieces. By forming an optimal problem for calculating the number of checksum pieces, we can achieve the best space efficiency in our approach. Furthermore, we divide the user data into pieces, and distribute them across multiple

cloud services; therefore, no single CSP can understand the uploaded user data. As a result, our approach can effectively protect user data from unauthorized access in the cloud, and provide security at the software level for the end users. Finally, the experimental results show that due to concurrent data processing, our approach provides very good performance in file uploading and downloading, with the cost of minor overhead for encoding and decoding data.

For future work, we will investigate possible ways to automatically select a suitable number of data pieces based on the network condition and the file size. We will consider other major aspects of cloud data, such as data integrity and confidentiality. For example, it would be feasible to adopt the digital signature technique to verify the integrity of the data stored in the cloud to ensure they were not altered by the service providers. Furthermore, when large cloud files are involved, the overhead for encoding and decoding may become a concern. To improve the overall performance in this case, we need to look into more advanced techniques for erasure codes, such as regenerating codes and non-MDS codes [3]. Finally, we will attempt to integrate our approach with cloud-based big data analysis for reliable and secure data stored in the cloud. This may also be considered as a worthy future direction.

REFERENCES

- [1] S. Yegulalp, "Adobe Creative Cloud Crash Shows that No Cloud is Too Big to Fail," *InfoWorld*, May 16, 2014. Retrieved on March 7, 2015 from <http://www.infoworld.com/article/2608200/cloud-computing/adobe-creative-cloud-crash-shows-that-no-cloud-is-too-big-to-fail.html>
- [2] C. Talbot, "Dropbox Outage Represents First Major Cloud Outage of 2013," *Talkin'Cloud*, Jan 15, 2013. Retrieved on May 18, 2014 from <http://talkincloud.com/cloud-storage/dropbox-outage-represents-first-major-cloud-outage-2013>
- [3] J. S. Plank, "Erasure Codes for Storage Systems: A Brief Primer," *Login: The USENIX Magazine*, www.usenix.org, December 2013, Vol. 38, No. 6, pp. 44-50.
- [4] C. Huang, H. Simitci, Y. Xu *et al.*, "Erasure Coding in Windows Azure Storage," *Proceedings of the 2012 USENIX Annual Technical Conference*, Boston, MA, USA, pp. 15-26, June 13-15, 2012.
- [5] L. B. Gomez, B. Nicolae, N. Maruyama, F. Cappello and S. Matsuoka, "Scalable Reed-Solomon-based Reliable Local Storage for HPC Applications on IaaS Clouds," *Proceedings of the 18th International Euro-Par Conference on Parallel Processing (Euro-Par'12)*, Rhodes, Greece, pp. 313-324, August 2012.
- [6] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads," *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST-2012)*, San Jose, CA, USA, pp. 20-33, February 2012.
- [7] N. Santos, K. Gummedi, and R. Rodrigues, "Towards Trusted Cloud Computing," *Proceedings of the Workshop on Hot Topics in Cloud Computing (HotCloud09)*, Article No. 3, San Diego, CA, June 15, 2009.
- [8] K. Hwang and D. Li, "Trusted Cloud Computing with Secure Resources and Data Coloring," *IEEE Internet Computing*, Vol. 14, No. 5, pp. 14-22, 2010.
- [9] D. Fitch and H. Xu, "A RAID-Based Secure and Fault-Tolerant Model for Cloud Information Storage," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 23, No. 5, 2013, pp. 627-654.
- [10] C. K. Clarke, "Reed-Solomon Error Correction," *R&D White Paper*, British Broadcasting Corporation, July 2002.
- [11] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, Amsterdam, London, New York, Tokyo, 1977.

Towards a Deployment System for Cloud Applications

Ruici Luo^{1,3}, Wei Ye^{2,3*}, and Shikun Zhang^{2,3}

¹School of Electronics Engineering and Computer Science, Peking University, China

²National Engineering Research Center for Software Engineering, Peking University, China

³Key Laboratory of High Confidence Software Technologies, Ministry of Education
{luoruici,wye,zhangsk}@pku.edu.cn

Abstract

A sophisticated deployment system plays an important role in automating and improving the process of software delivery, especially for cloud applications. Since cloud applications usually consist of many components run on different virtual machines, i.e., EC2 instances, the deployment is time-consuming and error-prone, which may involve manual operations and complex scripts. We develop a deployment system aiming to accelerate cloud application delivery. First of all, we propose a component model and a connector model involving cloud feature. Then we present a component management system, in which component can be configured and instantiated rapidly based component inheritance and composition. Finally, we develop a novel deployment mechanism that can automate deployment process across multiple cloud instances. Experiment shows that our approach can reduce the build time and downtime so that it can speed up the delivery process of software application.

Keywords: software architecture; application deployment; cloud computing; continuous delivery

1. Introduction

For enterprise applications, continuous integration is increasingly seen as an effective tool for reducing the cycle time from product backlog to receiving actual user feedback. This can result in real increases in developer and team productivity when combined with cloud computing. One key trend that is growing in importance daily is Continuous Delivery[13]. More and more organizations are looking to embrace an agile model in which stringent, auto-

mated testing allows enhancements or "micro-releases" to go live without the traditional waterfall release cycles. We are seeing a major shift in enterprise software development to cloud-based, continuous delivery, with fully automated quality, coverage, functional and performance tests gating live deployments. Thus, incremental deployment become more critical since it is very expensive to rebuild and re-deploy the whole application. Meanwhile, cloud applications usually consist of many components run on different virtual machines making deployment time-consuming and error-prone, which may involve manual operations and complex scripts. It has become a common issue to reduce the build time and downtime so that it can speed up the delivery process of cloud application.

In the past two decades, application servers have been designed to serve multiple applications, which means applications share identical runtime and deployment scenario. With the advent of Cloud Computing the IT resources can be rapidly and elastically delivered via Internet. Examples of compute Clouds are Amazons Elastic Compute Cloud (EC2) and Google App Engine (GAE). Meanwhile, the complexity of applications and servers grows rapidly. Nowadays, the benefits of making the server immutable[14] becomes more obvious and clear. If anything of an application has been changed, a new immutable server instance will be made next to the existing one, which will be destroyed soon. One can request and release resources in a few seconds with low costs, e.g. creating instances in EC2 and running applications in GAE. In this context, back-end infrastructure including middleware container has become one integral part of one specific application. LXC (Linux Containers) is an operating system level virtualization method for running multiple isolated Linux systems (containers) on a single control host, instead of creating a full-fledged virtual

*Corresponding Author

(DOI reference number: 10.18293/SEKE2015-192)

machine. We see that it has become feasible in cloud environment to make application deployment and runtime immutable with virtualization and lightweight container technology like Linux Container. If each component has its own infrastructure and isolated runtime, the cost of rebuild and redeploy a single component will be relatively low. By imposing a well-organized connectivity and lightweight communication mechanism between application components, breaking down partial components will not terminate the whole application. We could leverage the power of cloud infrastructure to support individual evolution, ensuring good replaceability and upgradeability of components in software system to achieve the goal of continuous delivery.

Application management is a key issue for successful continuous delivery. Developer need to take care of evolution of component configuration. Current solution for application management dedicated to IaaS(Infrastructure as a Service) and PaaS(Platform as a Service). Many configuration management systems such as Puppet[5], Chef[2], which provide a DSL to model a virtual machine instance, including files to present and application stack that should be running. These configuration management systems manage the configuration of applications in a centralized server and the work of deployment is assigned to operation teams, not developers. Although virtual machine[9] based on IaaS platform is a solution to deploy application, we also need a simple and lightweight way to manage the deployment of applications.

1.1. Contributions

This paper makes the following contributions:

- We present an component model from deployment perspective for accelerating continuous delivery process. In our approach, components consist of business function code, configuration options and runtime software stack(OS, middleware, dependencies library) definition which is called image. A image can be instantiated to a instance.
- We also provide a system to manage the images and instances. An inheritance mechanism ensures that each component can evolve independently and reused in many situations. Unlike all previous techniques and systems of which we are aware, our approach makes components immutable. If any changes occurs, a new image of the specific component is generated and instantiated to replace the old instance. Also the evolution process is recorded, i.e. the history of images is maintained by our deployment system. It is easy to rollback to any state of application.
- We present an system that automates the deployment of distributed application based on this model in the

cloud. The deployment system processes the instantiation and resolve the interconnections of components.

The rest of this paper is organized as follows. Section 2 presents a motivating example that illustrates how our approach works. Section 3 describes a component-based model and an application management system for cloud application. Section 4 describes an automated deployment system. Section 5 gives an example in practice to evaluate our approach. Section 6 discusses related works. We conclude in Section 7.

2. Component Model

To meet the challenges that applications in cloud should be highly scalable and flexible, we propose a component model as an abstraction of deployment elements. From the business function perspective, following the object-oriented principles of "Single Responsibility" and "Concerns Separation", a component should focus on a small, single and independent business function that it is responsible for. So parallel development can be organized straightforwardly and incremental evolution could be performed smoothly. From the infrastructure perspective, application changes are not only about application itself. Changes that is about environment should also be considered as part of evolution. Dynamic components interchangeability should be supported. So adding or updating components will not require redeployment of the entire application. Using the component model to abstract the target deploying application, we are aiming to following features:

- Self-contained infrastructure: Each Component contains the infrastructure required at the runtime. The infrastructure includes but is not limited to middleware and OS environments. This ensures the isolation of components and also improves the reliability since they do not affect each other.
- Individual Evolution: For the reason that each component has its own infrastructure and isolated to each other, they can be developed and maintained independently at the time. This can greatly improve the productivity of software and reduce the maintenance cost.
- Disposable Components: In cloud environment, virtualized instance can be acquired cheaply and pirated. The past method that deploying components into middleware or operating system is out of time.

We propose a component model, which is an extension of BU(Business Unit) model from BuOA[15]. BU contains presentation layer, business logic layer and data accessing layer inside. BU also provides attributes, operations and

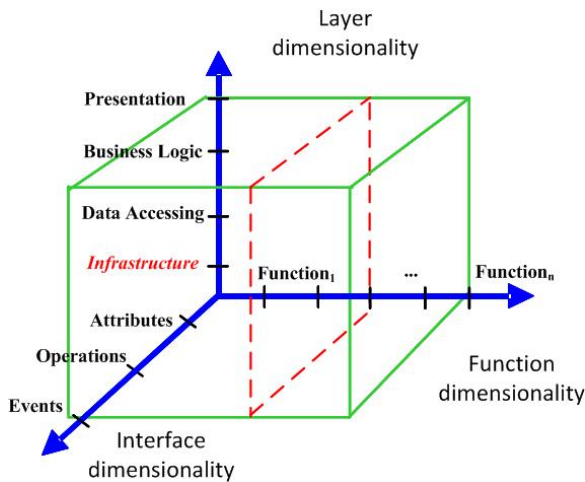


Figure 1. Extending infrastructure layer of BU

events as its external interface. Basically, attributes can be treated as representations of BUs internal state; operations provide ways to query and change its internal state; and events will indicate the changing of its internal state. However, BU only concerns abstraction in application logic level, lacking mechanism to support cloud features. Thus, we add infrastructure layer in Business Unit model, which includes middleware and OS environment. The extended model is illustrated in Figure 1.

The component model has two levels, business level and infrastructure level. Business level is not always necessary. A image with a MySQL database can be seen as having only infrastructure level. Component connectivity between images can be categorized into three categories base on the Two-Level perspective.

Business dependency This kind of relationship can be further divided into four categories: observing, injecting, weaving and binding as well as BuOA. We will not cover them in detail here.

Infrastructure dependency: This kind of relationship defines the dependencies of infrastructure, e.g. application server has dependencies of cache and database server.

Data sharing Another way to connect two components is to share data between them. An example is that multiple business components use a same database server.

3. Component Management

To support maintainance and evolution of components, we present a system to manage components of cloud applications.

3.1. Image

Image is subject to describe the state of a component, including hardware characteristics, software stack(OS, middleware, etc...) and applicative binaries. Each image has a unique identifier(usually generated by system via hash algorithms). There are many images generated in the evolution process of a component. So we aggregate these images to a *repository*. In analogy with Git version control system, a *image* is a commit and a *repository* is correspond to the same name. A *repository* potentially holds multiple variants of an image. In the case of our ubuntu image we can see multiple variants covering Ubuntu 10.04, 12.04, 12.10, 13.04, 13.10 and 14.04. Each variant is identified by a tag and you can refer to a tagged image like "ubuntu:14.04".

When developers decide to publish components for test or release, the management system just generate a new image with a unique identifier via the configuration about OS, middleware and applicative binaries and it is pushed to the central registry. The deployment system and other developers can pull this image and get an instance of it after pushing.

3.2. Instance

An *instance* is an instantiation of a *image* by assigning concrete values to configuration, and it is deployed to the IaaS platform as the runtime of a component. An *instance* consists of software stacks and applicative binaries. An instance has a global unique identifier as well as images. The identifier could distinguish two instances instantiated by one image.

3.3. Image Inheritance

To extend and reuse images for productivity, the image could be inherited. For example, each portion of a web application(web server, application server, cache, database, etc...) is running in a Ubuntu linux operating system. So we can make an "abstract"(which could also be instantiated) image for inheritance.

To extend base image, developers can add or override dependencies and configurations to generate new images which means that the configuration and the dependencies are all inherited from a base image. On the other hand, images with different versions of a same name should be co-located in one repository.

3.4. Image Composition

Another way to extend and reuse existing images is composition. For example, if we need a image that consists of

Java runtime environment and mysql database and there exist independent images of JRE and mysql. We can compose them and get a new image that has the java and mysql features.

However, it is not appropriate to merge images in some cases. If an image is based on "ubuntu" and another image is based on "windows", they can not be merged apparently. So before merge, we would check if the images has the same ancestor in the image tree, and then check if there are any conflicts between them. After composition the images tree becomes a Directed Acyclic Graph(DAG).

4. Application Deployment

4.1. Overview

To deploy the application to IaaS platform, we present a deployment system which takes a deploy plan as following configuration written by YAML:

```
web:
  image: onboard-core:1.2
  ports:
    - 8080
  volumes:
    - ./code
  links:
    - redis
redis:
  image: redis:latest
  command: redis-server --appendonly yes
```

This defines two components:

- **web**, which is built from an image called onboard-core with a version number 1.2. It also says to expose 8080 port, connect up the redis component and mount volumes for data sharing.
- **redis** which uses the redis image with latest version directly.

Each element on the top of the YAML file describes a component. It specifies the name and version of image and the dependencies to other component.

4.2. Disposable Distribution

As we mentioned above, to avoid the issue that infrastructure has been patched again and again, we make the infrastructure as part of application distribution. This means that any changes to the infrastructure is equivalent to the application. In our new situation, we absolutely know a system has been created via automation and never changed since the moment of creation. A distribution of application

is never modified after deployed, and merely thrown away after being replaced with a new distribution.

Another consideration is that the data related to an application is not immutable and cannot be thrown away. A practical way is shipping the data storage off of the BU distribution. Technically, sending log files to a central system log server, using shared file system like NFS, choosing mountable cloud service as storage devices are all feasible practice to guarantee data integrity.

4.3. Individual Evolution and Development

Incremental deployment is critical in the software evolution since it is very expensive to rebuild and redeploy the whole application. As we separate application into components each of which has its own infrastructure and isolated runtime, the cost of rebuild and redeploy a single component is relatively low. Due to the lightweight communication mechanism between components, breaking down partial BUs will not terminate the application. The individual evolution can also ensure good replaceability and upgradeability of components in software system.

To keep things simple, we consider two inter-related components in an application. One component requires services provide by another component and they are developed in parallel. As the developer(s) of each component, they do changes everyday with building and releasing SNAPSHOT version of distribution. So the developer(s) of the component that requires services of another does not need to get the source code and build another component, he/she/they only have to pull the SNAPSHOT of distribution and run it locally. This greatly reduce the time cost of dependencies building, testing and configuration. In summary, the collaboration mechanism between components varies from source code level to component with infrastructure level and will give a huge boost to improve the quality, reliability and productivity of software application.

5. Implementation and Evaluation

5.1. Component Implementation

We have implemented a prototype to verify our method and evaluate its performance. We use Docker[3] to implement the infrastructure level of components. The implementation is based on Spring Boot[6]. Spring Boot provides the ability to create stand-alone Spring based enterprise application that embeds an application as the middleware and can be run by itself. The prototype was tested on CentOS 6.4 and can also be applied or extended to the OS that supports Docker.

Application Logic Level of an image is the same as the architecture proposed in BuOA. In the example

the component projectMg contains three bundles projectMgt.persistence, projectMgt.service, projectMgt.web, corresponding data accessing layer, business logic layer and presentation layer respectively. They communicate with each other based on contracted service interfaces, shielding implementation details completely. For example, data accessing bundles can choose different Object-Relation mapping frameworks to do the persistence work as long as it keeps the data accessing interface unchanged.

Infrastructure Level of an image is implemented as a Docker container which contains a Spring Boot based application. The Docker container is a virtualized and isolated operating system, and the Spring Boot based application is embedded application server like Tomcat or Jetty. It is a stand-alone application which means no external server is required. To describe the Docker container, we add a Dockerfile to each BU. The Docker container is created via Dockerfile with application build. The following text file is an example description of infrastructure level of a image:

```
#Inherit from a built container
#with Java environment.
From komljen/jdk6-oracle
#Get the latest version of Maven
Run apt-get update
Run apt-get install -y maven
Run mvn clean install
#Startup the application
Cmd java -jar target/sample-1.0.0.jar
```

5.2. Evaluation

Onboard[4] is an actual application which continuous runs for about 2 year. To begin with, we develop with the BuOA approach. All BUs are put into a virgo instance. At that time, each BU only contains application code and is not isolated to each other. Any little change of a BU will cause the application to restart. More seriously, bugs in a BU cause the JVM process down and the application halts. The 10 components run in their own Docker container and are isolated to each other. There is one JVM process running as the runtime of each component. For the evolution perspective, each BU has its own individual evolutionary process.

The evaluation is based on the build time and downtime of each release. In the old architecture, the calculation is very easy because each build is related to the entire application and the downtime is the restart seconds of the application server. We have a continuous integration server to do daily release of application. We collect the logs from servers and make a table to show the data below (The data is up to June 2014).

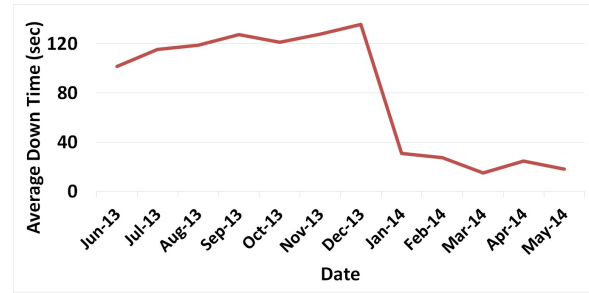


Figure 2. Downtime costs with the evolution of Onboard

Month	LOC	Change Times
July 2013	231137	4
Aug 2013	231251	5
Sep 2013	228879	3
Oct 2013	211297	4
Nov 2013	212132	6
Dec 2013	234853	4
The refactor separator		
Jan 2014	232268	16
Feb 2014	239247	19
Mar 2014	241317	20

Before June 2013, the application scale was relatively small, hence, newer data are shown in table above to keep our test in a consistent way. As we can see, the change times increased very fast when do the isolation of BUs. The reason is that each BU has its own evolution and the times is added by each of them. Developing with the new approach, the iteration has a much higher frequency.

The most important factor that affect the build time and downtime is the increment. With our new approach, increment of each release is quite small because upgrade a small part of BUs will not affect the status of other BUs. The test results are shown in Figure 2 and Figure 3. The data in these figures confirms that our new approach can reduce the build time and downtime so that it can speed up the delivery process of software application.

6. Related Works

The problem of deployment of application has attracted significant attention in the area of System Administration. Many tools exist: Puppet[5], Chef[2], CFEngine[1]. The goal of these systems is to simplify the management task of large scale machines. However, they only consider the configuration of systems or environments and not take the configuration and interconnections of components in application.

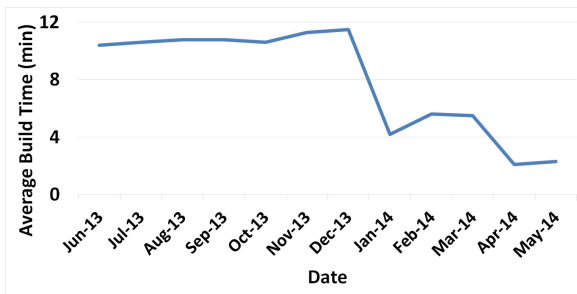


Figure 3. Build time costs with the evolution of Onboard

Aeolus[7] component model is specifically designed to capture realistic scenarios arising when configuring and deploying distributed applications in cloud environments. It is able to describe several component characteristics such as dependencies, conflicts, non-functional requirements. The Blender[10] toolchain extends [8] that automates the assembly and deployment of complex component-based software systems. By relying on a configuration optimizer and a deployment planner, the final deployment satisfies not only user requirements but also to be optimal with respect to the number of used virtual machines.

Engage[11] is a deployment management system. Throughout the paper the term *resource* is used as a synonym of component. *Resource* consists of *type* and *driver*. The former statically verifies deployment properties and generates the deployment plan, while the latter installs and manages the resource's lifecycle. Engage introduces three types of dependencies: **Inside** for nesting (e.g. application code runs into an application server); **Env** for local dependencies (Java programs need JRE); **Peer** for resources deployed anywhere else. The present paper has a similar idea to Engage: It separates the specification and runtime of components and automated generates the right order of deployment.

SmartFog[12] is a Java framework to manage deployment for distributed applications. It shares some concepts with the Engage that each component has a declarative description and a driver called lifecycle manager.

7. Conclusion

To address the critical challenge of deploying distributed application in the cloud, we present an component-based model that aims to automated configure and deploy over lightweight container. Base on the proposed model, we introduce an application management system as well as an inheritance-based mechanism that ensures each resource can be evolved independently and reused in different scenarios. We also present a deployment system that could

process the dependencies and interconnected relationship of components automatically. To evaluate our approach, we implement a distributed application on an industrial IaaS platform. As a result, we can decompose a cloud application vertically into independent and cohesive modules which has dynamic interchangeability and evolvability.

Acknowledgments: This work was supported by the National Natural Science Foundation of China under Grant No.61202070.

References

- [1] Cfengine. <http://cfengine.com/>. Accessed: 2015-05-10.
- [2] Chef. <https://www.chef.io/>. Accessed: 2015-05-10.
- [3] Docker. <https://docker.com/>. Accessed: 2015-05-10.
- [4] Onboard. <https://onboard.cn/>. Accessed: 2015-05-10.
- [5] Puppet. <https://puppetlabs.com/>. Accessed: 2015-05-10.
- [6] Spring boot. <http://projects.spring.io/spring-boot/>. Accessed: 2015-05-10.
- [7] M. Catan, R. D. Cosmo, A. Eiche, T. A. Lascu, M. Lienhardt, J. Mauro, R. Treinen, S. Zacchiroli, G. Zavattaro, and J. Zwolakowski. Aeolus: Mastering the Complexity of Cloud Application Deployment. In K.-K. Lau, W. Lamersdorf, and E. Pimentel, editors, *ESOC - European Conference on Service-Oriented and Cloud Computing - 2013*, volume 8135, pages 1–3, Malaga, Spain, 2013. Springer.
- [8] R. Di Cosmo, M. Lienhardt, R. Treinen, S. Zacchiroli, J. Zwolakowski, A. Eiche, and A. Agahi. Automated synthesis and deployment of cloud applications. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14*, pages 211–222, New York, NY, USA, 2014. ACM.
- [9] X. Etchevers, T. Coupaye, F. Boyer, and N. de Palma. Self-configuration of distributed applications in the cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 668–675, July 2011.
- [10] X. Etchevers, G. Salaün, F. Boyer, T. Coupaye, and N. De Palma. Reliable self-deployment of cloud applications. In *SAC 2014 - 29th ACM Symposium on Applied Computing*, Gyeongju, South Korea, Mar. 2014.
- [11] J. Fischer, R. Majumdar, and S. Esmailsabzali. Engage: A deployment management system. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12*, pages 263–274, New York, NY, USA, 2012. ACM.
- [12] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft. The smartfrog configuration management framework. *ACM SIGOPS Operating Systems Review*, 43(1):16–25, 2009.
- [13] J. Humble and D. Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.
- [14] K. Morris. Immutable server, June 2013.
- [15] W. Ye, R. Luo, S. Zhang, X. Liu, and W. Hu. Buoa: An architecture style for modular web applications. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 802–807. IEEE, 2012.

Impact of Unanticipated software evolution on development cost and quality: an empirical evaluation

Rodrigo Vilar

Exact Sciences Department
Federal University of Paraíba
Rio Tinto, Brazil
rodrigovilar@dce.ufpb.br

Anderson Lima, Hyggo Almeida, Angelo Perkusich
Embedded Systems and Pervasive Comp. Lab.
Federal University of Campina Grande
Campina Grande, Brazil
anderson.lima, hyggo, perkusic@embedded.ufcg.edu.br

Abstract—Most techniques to aid maintenance and evolution of software require to define extension points. Generally, developers try to anticipate the parts that are more likely to evolve, but they can make mistakes and spend money in vain. With Unanticipated Software Evolution, developers can easily change any element of the software, even those that are not related with an extension point. However, we have not found empirical validations of Unanticipated Software Evolution impact on development cost and quality. In this work, we design and execute an experiment for Unanticipated Software Evolution (specifically, using the COMPOR platform), in order to compare its results metrics -- time, lines of code, test coverage and complexity -- using OO systems as baseline. 30 undergraduate students were subjects in this experiment. We concluded that COMPOR have significant impact on the Lines of code and Complexity metrics, reducing the amount of lines changed and the McCabe cyclomatic complexity on evolution of a small system.

Keywords—Unanticipated Software Evolution, Cost, Quality, Empirical software engineering, Software Evolution.

I. INTRODUCTION

Some studies estimate that maintenance and evolution tasks spend between 50% and 90% of software development budget [13, 8]. Thus, Software Engineering researchers invest considerable resources in order to create new techniques that ease and reduce the cost of software evolution. Most of these techniques require developers to anticipate extension points (EP), which are flexible structures to hold new functionality and changes.

However, there is a trade-off: defining an EP is abstract and expensive; conversely, it is even more expensive to change software pieces that are not prepared for it. So, for each EP created, we expect a ROI (return of investment), deriving out of reducing the cost of later changes that use the same EP. For this reason, developers try to discern and isolate software chunks inclined toward change. Nevertheless, sometimes they do not predict EP correctly and ROI is zero.

Unanticipated Software Evolution (USE) is a Software Engineering approach, which aids developers to change any software fragment, even without EP [12]. It considers that is possible to reduce the cost of software evolution and preserve its quality, even when there is not investment to create EP. As a

result, it would eliminate the trade-off we cited above and developers would not worry to create EP.

In an effort to confirm USE hypothesis, we have analyzed all articles published on USE events [12, 10, 11]. Nevertheless, none of these studies have validated the influence of USE on software development metrics such as quality and cost. In face of this gap, we define a business problem for this work.

Business Problem: *There is no convincing evidence on how USE influences software development metrics.*

In this paper, we perform an early evaluation of COMPOR [5, 6], a USE platform developed by Embedded Laboratory at UFCG¹, whose code is open source and is available online². COMPOR is a container for components that communicate with each other indirectly, through a specific message mechanism. So that components have weak coupling and can be easily changed. In fact, COMPOR can even change components at run time.

We have defined a technical problem, reducing our scope to COMPOR and using more specific software development metrics.

Technical Problem: *There are not empirical studies that investigate COMPOR influence on software development cost and quality.*

We propose an experiment to fulfill this gap on USE validation. Since COMPOR is a USE platform, its experimental outputs are also USE results. So, we try to evaluate USE impact over software development through COMPOR.

Cost and Quality are abstract metrics. So we choose concrete metrics for our experiment. We measure Cost as the time spent and lines of code changed in order to complete a software evolution task. Likewise, we assess Quality being Cyclomatic complexity and Test coverage of code after evolution.

In an ideal configuration, the experiment should use professional developers to implement systems with two alternatives – coding using only Object oriented code or using

¹ <http://www.embeddedlab.org>

² <http://bit.ly/COMPOR>

COMPOR – and compare the Time, LoC, Complexity and Coverage results. This way we would infer COMPOR impact on software development, considering OO results as baseline.

Due to resources and time limitations, we performed our experiment with undergraduate students, during an OO Design course. Carver et. al. [3] state that the risk of using inexperienced students is justifiable for pilot experiments. This kind of experiment would not be generalizable, but it contributes to fix experiment design problems and to guide future replications on professional development environments.

At this point, we can define our objective and hypotheses using the QM template.

Objective: The purpose of this study is to measure the impact of COMPOR on evolution cost (time spent and lines of code changed) and quality (tests coverage and complexity) [16], from the point of view of software developers, in the context of evolution tasks for a small system implemented by undergraduate students, using plain Object Oriented implementations as baseline.

Hypothesis 1: COMPOR systems require less time to complete evolution tasks than plain OO systems;

Hypothesis 2: COMPOR systems change less lines of code to complete evolution tasks than plain OO systems;

Hypothesis 3: COMPOR systems have better test code coverage after evolution tasks than plain OO systems;

Hypothesis 4: COMPOR systems have lower cyclomatic complexity after evolution tasks than plain OO systems.

In the remaining of this paper, we show the related work, describe COMPOR features, detail the experiment design, analyze the experiment results and point out our conclusions and future work.

II. RELATED WORK

We have divided this section into two parts. Firstly, we review the literature about USE, looking for concrete tools and their empirical validation. After that, we show some experimental works for software evolution, similar to our experiment.

A. Unanticipated software evolution

We have found some USE works in literature. Oreizy et. al. defined an architecture for run-time software evolution [14]. Keeney and Cahill created a framework for dynamic adaptation [9]. Wurthinger et. al. modified a Java virtual machine to allow arbitrary runtime changes at any point at which a Java program can be suspended [19]. Piechnick et. al. propose a role-based composition system that enables the adjustment of unanticipated, dynamic self-variation of applications in a fine-grained manner [15]. However these works did not evaluate empirically the impact of USE on software cost and quality. Therefore, we expanded the scope of our literature review to experimental evaluation of software evolution.

B. Software evolution experiments

While investigating software evolution literature, we found several experimental studies that evaluated aspects of evolution.

Arisholm and his partners worked three times with alternative designs for a coffee machine simulator, which sells and prepares drinks [4], in order to: evaluate changeability of systems with good and bad design [2]; measure the effect of sequence in which maintenance tasks are performed on the time required to perform them and on the functional correctness of the changes made [18]; evaluate the effect of centralized versus delegated control design on software maintainability [1].

Deligiannis et. al. have replicated the [1] study, using other programming language, enhancing the evolution tasks and collecting more metrics [7]. Sfetsos et. al. used the coffee machine project to investigate the impact of developer personalities and temperaments on communication, pair performance and pair viability-collaboration [17].

In spite of not focusing on USE, these works helped us to design a comparative experiment for COMPOR.

III. COMPOR

This section explains the Unanticipated Software Evolution features of COMPOR that we evaluated in an experiment. COMPOR has a generic and formal specification for a component container. It defines the components structure and their indirect communication, decoupling components and easing their change. For example, in Fig. 1, a component A needs to invoke a service of another component B, A does not invoke B directly. Instead, B should declare a service named as *s* and A could use the COMPOR API to invoke service *s*. The COMPOR container discovers automatically where *s* is declared and invokes it.

In a hypothetical evolution scenario, the system client requires to change some functionality of *s*. The system developer only needs to deploy another component C that also declares a *s* service, replacing the former service from B. A and other components that invoke *s* are not aware of that change, since they do not know B and C components directly.

Currently, there are four COMPOR implementations, for Java, C#, C++ and Python languages. In this experiment, we used the Java Component Framework for COMPOR, which defines two API classes: `ComporFacade`, that must be extended to create the system entry point and to deploy components; and `Component`, that must be extended too, in order to declare services and invoke services of other components.

IV. EXPERIMENT DESIGN

A. Experimental Units

With a view to run this experiment using evolution tasks, we needed to choose a system to be implemented by the experiment subjects. We decided to use a small system, which can be completely implemented by just one developer, rather than a big one that demands several developers working together.

The coffee machine problem fits this small-system requirement. It has also been replicated on several experimental studies. So, we have decided to use the same project (with some adaptations) for this experiment.

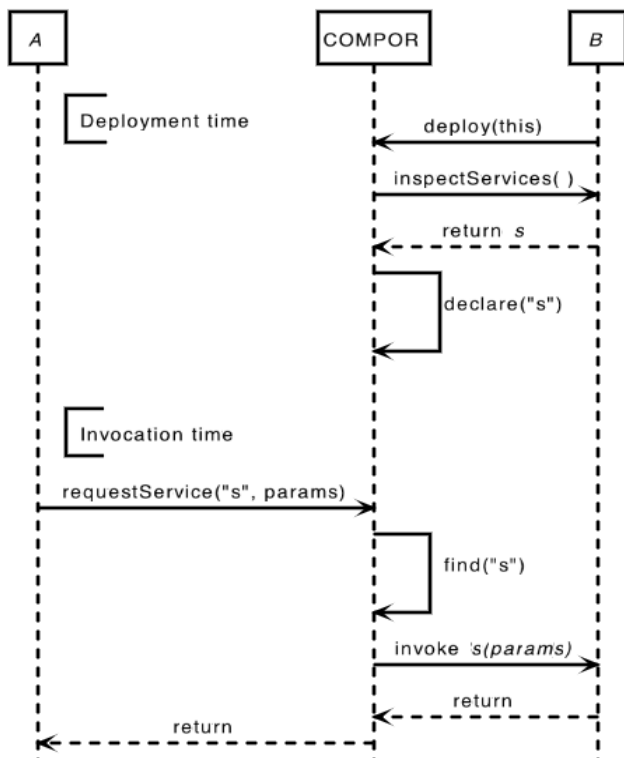


Figure 1. COMPOR: Declaring and invoking.

We planned its development as evolution tasks, which are the Experimental Units of our experiment.

The original coffee machine problem has four phases [4]: Payment with coins, four types of drink with the same price, cancel drink and return coins; add a new drink type with another price; use employees badges to directly debit the cost of drink purchases from paychecks; dynamic drink configuration.

We have partitioned these phases into 50 small tasks, so that developers can achieve better success rates on evolution tasks. Among these tasks there are also some new tasks that we have added to fill some missing functionality, e. g., loading coins on machine start in order to provide change.

Since COMPOR is a tool that starts operating from software design, the experiment does not need to measure COMPOR influence on requirements and analysis development phases. Therefore, we have simulated that requirements and analysis phases were already finished, and provided automatic functional tests for each evolution task. The functional tests run against a specific coffee machine Facade, which can be implemented using COMPOR or plain object orientation.

Next subsections detail the treatments designed to run with the experimental units.

B. Input: Independent variables

Factor: Technology

The main focus of this experiment is to compare result metrics of evolution tasks, between implementations that used COMPOR versus other versions that used plain OO. Therefore,

the experiment contains a simple design with only one interesting factor, which has two levels: using COMPOR or using plain OO. The other sources of variation are undesired. So, we have designed the experiment in order to neutralize their effect over dependent variables.

Undesired controlled variable: Participants

The experiment engaged 30 third-year students of Licentiate in Computer Sciences at UFPB, which were taking an OO Design course. The OO Design teacher used this experiment to grade the students in a practical project. Each student has different levels of experience in OO programming. While some of them work as junior developers on start-ups, others have almost no programming skills. Before the experiment, all students already had classes of refactoring techniques and design patterns.

To reduce the effect of developer experience, we have allocated them into random pairs. In fact, when we compared the final grading score in OO Design class, the score standard deviation for individuals was 0.89. In other hand, the score standard deviation for the experiment pairs was 0.61. Therefore, we suggest that pair randomization really reduced the developer experience effect. Each pair performed pair programming during the evolution tasks. Moreover, the experiment design uses replication for participants, because each pair should carry out all evolution tasks. We divided the 15 student pairs randomly into two groups. The first five pairs should use COMPOR in the coffee machine implementation and the other ten pairs must use plain OO.

Undesired not controlled variable: Environment

There are some events that we cannot control and would impact the experiment results, such as, climate, holidays, students transportation problems, etc. In order to reduce the environment effect, we have designed the experiment in a controlled manner. All students worked in a laboratory at UFPB with similar schedule, resources and instructions to execute

C. Procedures

Preparing

We have prepared some guidelines to guide students through experiment:

- An Experiment Manual³, explaining experiment conditions, purpose, resources, steps, auxiliary documentation and glossary;
- A Web Form to manually collect experimental unit configuration and time spent metric;
- Auxiliary documentation containing pseudo code, because instead of evaluating algorithms, we want to analyze design decisions;
- A Github repository for a coffee machine specification, containing a sequence of 50 tags (one for each evolution task). Each tag defines the functional tests,

³ <http://bit.ly/CoffeeMachineExperiment>

using jUnit⁴ and Mockito⁵, for its respective evolution task;

- A tutorial which we have used to give a class about COMPOR;
- A Github repository with the last COMPOR version.

The students were also trained on: Git, to manipulate Github repositories; Maven, to manage projects dependencies; JUnit and Mockito, to understand and execute the functional tests; Facade design pattern, which was used by the functional tests.

Executing

In the Coffee machine Github repository, we have created a tag for each evolution task, with an X.YY format. Where X means the coffee machine phase (from 1 to 4) and YY is the evolution task inside of the phase.

Beginning on tag 1.01 until tag 4.15, the student pair has to follow this procedure:

- a) Merge the current implementation code with the next tag (except for tag 1.01, which has not implementation);
- b) Set the task start time;
- c) Evolve the code until all functional tests pass;
- d) Set the task finish time;
- e) Commit the task final code and send it to Github;

Submit the Web Form with task data, such as start and finish times, pair id, task id, technology and subjective questions about difficulties.

The results of first five evolution tasks were fragile, since we consider it as training for experiment *modus operandi*.

D. Output: Dependent Variables

After the Executing procedure detailed above, we can collect several data about each evolution task. Github provides a diff report for each commit, so we can calculate the amount of lines of code changed by the evolution task. The time spent is collected from a spreadsheet populated by the web form.

Collecting Complexity and Coverage data is harder, since we need to access each evolution task final code and run the Cobertura Maven plugin⁶. It generates an HTML report with total test code coverage and mean McCabe cyclomatic complexity.

V. ANALYSIS

In this section, we show the experiment result data and its transformations, in order to try to obtain normal-distributed data. We also perform some statistical tests and interpret the models results. After all, we check the hypotheses defined in Section 1.

A. Results and Transformations

The initial 26 experiment tasks (1.01 to 1.26) represent the first coffee machine requirement. While the subsequent ones represent evolution tasks. Due to software evolution

importance and COMPOR evolution nature, we focused our analysis on evolution tasks (2.01 to 4.15).

Each student pair worked 35 hours in this experiment, but only one pair finished all evolution tasks successfully. All teams submitted approximately 500 experimental task logs.

We have ignored some data due to the following reasons:

- After task 4.01 there is not enough data to perform statistical tests;
- Three COMPOR teams used COMPOR poorly. Their Facades are replete of OO code and invoke COMPOR only three times. By comparison, the two remaining teams have smaller Facades and invoke COMPOR 8 and 23 times, respectively;
- The 3.02 Task alone weakened all COMPOR metrics. Since, in the 3.03 task, the metrics returned to normal levels, we consider the former task as an outlier.

This resulted in 338 observations that we have analyzed using the R statistical language. The tasks data, R code and program output are available online⁷.

In the scope of this experiment, tasks size vary a lot and absolute data for response variables did not tend to be normally distributed. With this in mind, we have transformed raw experiment data into relative values based on the mean OO metrics for each task. For example, the mean Time for all OO teams on task 1.01 was 35 minutes. So, instead of using the absolute Time value for Team 01 on this task (106 minutes), we have made statistical tests using the respective relative value (302% of OO mean). The relative data became closer to the normal distribution than the absolute data.

After that, we applied a log transformation into Time and LOC metrics and they become almost normally distributed. We did not find any transformation that made Coverage and Complexity data normal. So, these variables were tested with non-parametric methods.

Since the experiment generated a lot of observations, we have decided to split the observations into seven sequential task groups of about 50 observations. After that, we made separated statistical tests for each task group. Therefore, we compared OO and COMPOR metrics seven times during experiment execution and got temporal conclusions for experiment results.

The task groups had different configurations for each response variable (Time, LOC, Coverage and Complexity). We made some group adjustments in order to find group boundaries where response variables change behavior. This approach optimized the results of statistical tests.

Firstly, we tested the normality and homoscedasticity of data for each combination of task group and response variable, for both OO and COMPOR teams. In the sequence of analysis, we used t tests for normal data and Wilcox tests for non-normal data, in an effort to discover significant relations between COMPOR and OO data: Do COMPOR metrics differ of OO metrics? Are COMPOR metrics lower than OO metrics? And

⁴ <http://www.junit.org>

⁵ <http://code.google.com/p/mockito/>

⁶ cobertura.github.io/cobertura/

⁷ <http://bit.ly/Comporexperimentresults1>

are COMPOR metrics greater than OO metrics? At least, we performed Power tests to analyze the probability of type II errors.

B. Interpretation

The relative Time spend to complete tasks (Table 1) had an alternating behavior in the experiment beginning. Between 1.04 and 1.24 tasks, 12 tasks spent less time with OO and 9 tasks spent fewer time with COMPOR. These results have considerable significance ($p\text{-value} < 0.03$) and power above 0.7.

As the experiment reached evolution tasks, COMPOR and OO metrics equalized. In spite of the low statistical power, this data indicates the COMPOR Time spent for evolution tasks is better than its own Time for development tasks. We should replicate this experiment, in order to obtain sufficient data until the 4.15 task and analyze the metrics trends. There still is one question: will COMPOR Time tend to equate OO Time infinitely or COMPOR will overcome OO?

In relation to the LOC metric, the relative amount of lines changed is equal for both technologies until task 1.25. The last 9 (evolution) tasks demanded less lines for COMPOR versions. As Table 1 shows, these data is significant and has statistical power above 0.5.

Regarding Test Coverage, we gave equal and fixed tests, mapping each task requirements, for all teams. So, low coverage rates mean that a team created a lot of unnecessary code, which reduces the code quality.

COMPOR teams had better coverage in the first 11 tasks and equals coverage in the middle 23 tasks. However, in the last 4 tasks, OO teams got better coverage results. This means that COMPOR teams wrote more unnecessary code in the

experiment end and the use of COMPOR would impact system quality.

Finally, we analyzed the Complexity metric, where the first 15 tasks had similar results for both technologies. In the 11 intermediary tasks, the COMPOR teams code complexity was significantly lower than OO code. The last 8 (evolution) tasks showed similar results again, but with low statistic power. This means that there is a great probability that the statistical tests, which are not significant for tasks 2.01 - 4.01, were wrong. This data needs more replication to enhance the last tests power and find out Complexity trends: does COMPOR continue to generate code with lower complexity as system increases? This answer can be found in a future work.

C. Hypothesis test

Hypothesis 1: Does COMPOR systems require less time to complete evolution task than plain OO systems?

There is no significant trend on the impact of Technology factor on the Time teams took to perform the evolution tasks. So, we REJECT this hypothesis.

Hypothesis 2: Does COMPOR systems require less lines of code to complete evolution task than plain OO systems?

COMPOR teams changed less lines of code to implement evolution tasks. So, we ACCEPT this hypothesis.

Hypothesis 3: Does COMPOR systems have better test code coverage after completing evolution task than plain OO systems?

For almost all tasks, the Technology factor did not have significant impact on test coverage after evolution tasks, in the COMPOR versus OO comparison.

TABLE I. STATISTICAL TESTS FOR TIME, LOC, COVERAGE AND COMPLEXITY RESPONSES VARIABLES.

Variable	Task group	Normal data	Equal variance	Statistical tests (p-value)			Statistical power
				COMPOR != 0	COMPOR < 0	COMPOR > 0	
Time	1.04 - 1.07	Yes	Yes	0.042	0.021	NS	0.78
	1.08 - 1.11	Yes	Yes	0.003	NS	0.001	0.96
	1.12 - 1.16	Yes	Yes	0.048	0.024	NS	0.73
	1.17 - 1.24	Yes	Yes	0.054	NS	0.027	0.75
LOC	1.26 - 3.01	No	NA	0.053	0.026	NS	0.51
	3.03 - 4.01	Yes	Yes	NS	0.064	NS	0.59
Coverage	1.01 - 1.04	No	NA	0.058	NS	0.029	0.74
	1.05 - 1.07	Yes	No	0.087	NS	0.043	0.46
	1.08 - 1.11	No	NA	NS	NS	0.091	0.63
	3.03 - 4.01	No	NA	0.016	0.008	NS	0.999
Comp.	1.16 - 1.20	No	NA	0.044	0.022	NS	0.999
	1.21 - 1.26	No	NA	0.021	0.01	NS	0.99

a. This table shows only significant statistical results.

Moreover, in the last evolution tasks, OO teams wrote code with better coverage than COMPOR ones. So, we REJECT this hypothesis.

Hypothesis 4: Does COMPOR systems have lower cyclomatic complexity after completing evolution task than plain OO systems?

In spite of the final evolution tasks inconclusive results, COMPOR teams produced less complex code for 11 intermediary tasks. Therefore, we ACCEPT this hypothesis.

D. Threats to validity

Conclusion validity

Due to the low experience of the experiment subjects, we have noticed that most pairs did not take care of system design. They just want to finish the most tasks possible. These groups have written bad code and randomly made some refactoring, which could have influenced some results. We suggest, in future replications of this experiment, to reserve a period of time, after each task, only to perform refactoring.

Random irrelevancies can affect results, such as lack of internet, hardware problems on PCs, etc. Another threat is related to human experimental subjects who can easily change their behavior over time, generating noise to the data.

External validity

The main threat to validity in this experiment is to generalize the results, from the sample we have chosen – undergraduate students – to the real population of programmers. In the context of a company, it is expected that employees have a reasonable leveling in relation to software development. On the other hand, the academic environment is very heterogeneous, in terms of ability and knowledge.

Another problem is the generalization of COMPOR results to the whole class of Unanticipated Software Evolution tools, because COMPOR good metrics could be result of another COMPOR characteristic apart from USE. With this in mind, we hid most COMPOR function, such as, run-time adaptation and exposed only the inter-component communication.

VI. CONCLUSIONS

In this work, we designed and executed the first experiment, as far as we know, for Unanticipated Software Evolution that measures the effects of this technique on development cost and quality.

We consider the experiment design as a valid contribution, because it can be easily replicated with better configurations in order to obtain more relevant results. In the scope of this work, we have obtained some significant results for USE influence on development cost and code quality. While COMPOR significantly reduces the amount of lines changed and code complexity, it does not affect development time and code test coverage.

As future works, we suggest the replication of this experiment with other configurations: providing with sufficient time to complete all evolution tasks, until task 4.15; inviting professionals to perform evolution tasks and given them better COMPOR training; and using other USE tools. After acquiring

stronger evidences of USE hypothesis, other works may also measure COMPOR performance overhead.

REFERENCES

- [1] E. Arisholm and D. I. Sjöberg. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *Software Engineering, IEEE Transactions on*, 30(8):521–534, 2004.
- [2] E. Arisholm, D. I. Sjöberg, and M. Jørgensen. Assessing the changeability of two object-oriented design alternatives—a controlled experiment. *Empirical Software Engineering*, 6(3):231–277, 2001.
- [3] J. Carver, L. Jaccheri, R. Morasca, and F. Shull. Issues in using students in empirical studies in software engineering education. In *IEEE METRICS*, page 239. Prentice Hall, 2003.
- [4] A. Cockburn. The coffee machine design problem: Part 1 & 2. *C/C++ Users Journal*, may/june 1998.
- [5] H. de Almeida, A. Perkusich, E. Costa, and R. Paes. Compore: a methodology, a component model, a component based framework and tools to build multiagent systems. *CLEI Electronic Journal*, 7(1), 2004.
- [6] H. O. de Almeida, A. Perkusich, G. Ferreira, E. Loureiro, and E. de Barros Costa. A component model to support dynamic unanticipated software evolution. In *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2006)*, San Francisco, CA, USA, July 5-7, 2006, pages 262–267, 2006.
- [7] I. Deligiannis, P. Sftesos, I. Stamelos, L. Angelis, A. Xatzigeorgiou, and P. Katsaros. Assessing the modifiability of two object-oriented design alternatives— a controlled experiment replication. In *Proceedings 5th EUROSIM Congress on Modelling and Simulation*, 2004.
- [8] L. Erlikh. Leveraging legacy system dollars for ebusiness. *IT professional*, 2(3):17–23, 2000.
- [9] J. Keeney and V. Cahill. *Chisel: A policy-driven, context-aware, dynamic adaptation framework*, 2003.
- [10] G. Kniessel, P. Costanza, and J. L. Fiadeiro. Second international workshop on unanticipated software evolution, Apr. 2003.
- [11] G. Kniessel and T. Mens. First international workshop on foundations of unanticipated software evolution, Mar. 2004.
- [12] G. Kniessel, J. Noppen, T. Mens, and J. Buckley. First international workshop on unanticipated software evolution, June 2002.
- [13] B. P. Lientz and E. B. Swanson. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.
- [14] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In *Proceedings of the 20th International Conference on Software Engineering*, pages 177–186. IEEE Computer Society, 1998.
- [15] C. Piechnick, S. Richly, S. Gotz, C. Wilke, and U. Aßmann. Using role-based composition to support unanticipated, dynamic adaptation smart application grids. In *ADAPTIVE 2012, The Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications*, pages 93–102, 2012.
- [16] D. Racadon. *Developers’ seven deadly sins*, July 2014.
- [17] P. Sftesos, I. Stamelos, L. Angelis, and I. Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. *Empirical Software Engineering*, 14(2):187–226, 2009.
- [18] A. I. Wang and E. Arisholm. The effect of task order on the maintainability of object-oriented software. *Information and Software Technology*, 51(2):293–305, 2009.
- [19] T. Wurthinger, C. Wimmer, and L. Stadler. Unrestricted and safe dynamic code evolution for java. *Science of Computer Programming*, 7 2011.

An empirical study on the impact of Python dynamic features on change-proneness

Beibei Wang, Lin Chen^{*}, Wanwangying Ma, Zhifei Chen, Baowen Xu

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Abstract—The dynamic features of programming languages are useful constructs that bring developers convenience and flexibility, but they are also perceived to lead to difficulties in software maintenance. Figuring out whether the use of dynamic features affects maintenance is significant for both researchers and practitioners, yet little work has been done to investigate it. In this paper, we conduct an empirical study to explore whether program source code files using dynamic features are more change-prone and whether particular categories of dynamic features are more correlated to change-proneness than others. To this end, we statically analyze historical data from 4 to 7 years of the development of seven open-source systems. We employ Fisher and Mann-Whitney hypothetical test methods, along with logistic regression model to solve three research questions. The results show that: (1) files with dynamic features are more change-prone, (2) files with a higher number of dynamic features are more change-prone, and (3) *Introspection* is shown to be more correlated to change-proneness than the other three categories in most systems. This innovative work can give some inspirations and references to researchers who are always focusing their eyes on how and why the dynamic features are used. For practitioners, we suggest them to be wary of files with dynamic features because they are more likely to be the subject of their maintenance effort.

Keywords- *dynamic features; change-proneness; Python; empirical software engineering; open-source*

I. INTRODUCTION

In recent years, many researchers have shown great interest in the use of dynamic features or dynamic behaviors of programming languages, such as Python, JavaScript and Ruby. Previous works were conducted mainly to discuss whether practitioners are willing to use dynamic features, the main reasons that drive people to use them and how these features are used [1], [2], [3], [4]. Besides, there is a long and ongoing debate about the possible pros and cons of dynamic features in programming languages. Some authors state that dynamic features are of benefit for their flexibility, expressivity and succinctness [5]. For example, the commonly available reflective mechanisms include support for checking available fields/methods, adding and removing fields/methods without the need to restart or rebuild the running program. Others hold the opposite view that the use of these features may hinder software evolution and lead to difficulties in software maintenance. For instance, the use of *eval* endows programmers with the ability to extend applications, at any time, and in almost any way they choose, but it will affect the optimizations that can be applied to programs and significantly limit the kinds of errors that can be caught statically and the security guarantees that can be enforced [4]. Hence, it is of great significance to investigate the relation between the use of dynamic features and system maintenance. However, to the best of our knowledge, little work

was focused on the effect of dynamic features on program maintenance or evolution, let alone the use of Python dynamic features. Therefore, we make an empirical study on the relation between dynamic features and change-proneness which is well-known to be an indicator of maintenance in the previous study.

Goal. We aim to investigate the effects of 18 Python built-in dynamic features, classified into four broad categories, on the three types of code evolution phenomena. First, we study whether files with dynamic features have an increased likelihood of changing compared to other files. Second, we study whether files with more dynamic features than others are more change-prone. Third, we study the relation between the particular categories of dynamic features and change-proneness.

Contribution. This paper makes three contributions.

- This work is the first one to consider the effect of dynamic features on change-proneness, especially concerning Python language, and thus it will give some inspirations and references for the successors.
- We analyze multiple historical releases of 7 open-source systems to collect the occurrence of 18 Python built-in dynamic features of each file and change information between two versions. The data we gather and publish¹ are useful for the follow-up studies related.
- We get an instructive conclusion from the results of the experiment that although developers are benefit from the flexibility and convenience brought by dynamic features, they should be prudent with them since these features might contribute to more maintenance effort.

The remainder of this article is structured as follows. Section II introduces an overview of related work. Section III provides a description of the 18 Python dynamic features as well as the classification and our detection approach for them. Section IV describes the exploratory study definition and design. Section V presents the study results. Section VI gives a detailed explanation and discussion, along with threats to validity. Finally, Section VII concludes the study and outlines the future work.

II. RELATED WORK

Until now, as far as we know, there has been no study of the relation between dynamic features and change-proneness. Several works studied the usage of dynamic features of various languages, such as JavaScript, Smalltalk and Python, by dynamically or statically analyzing the source code. We will summarize these works as well as works that aimed at relating software quality with factors such as metrics, code smells and language characteristics.

^{*}Corresponding author: Lin Chen; E-mail: lchen@nju.edu.cn

¹<https://github.com/MG1333051/Detailed-Research-Results-git>

Previous research on the dynamic features concerned how to collect them and why and how these features were used in practice. Callaú et al. [1] studied the reflection feature in Smalltalk and found that if a large portion of the usages of dynamic features cannot be refactored, others work around limitations of the programming languages. Richards et al. [4] performed a large-scale study on the use of *eval*, the result of which showed that *eval* was often misused and many uses were unnecessary and could be replaced with equivalent and safer code. Holkner and Harland [7] have conducted a study of the use of 14 dynamic features in the Python programming language. Their study focused on a smaller set of programs and concluded that dynamic features occur mostly in the initialization phase of programs and less so during the main computation. Further, Åkerblom et al. [5] did a similar research to Holkner’s study. They showed that dynamic behaviour is neither buried in library code, nor predominantly occurs at program startup time, which is in slight contrast to the results of Holkner’s study. In our study, we were partly inspired by their classification of dynamic features.

Some studies used metrics as quality indicators, such as Basili et al.’s seminal work [9]. Cartwright and Shepperd [10] performed an empirical study on an industrial C++ system, supporting the hypothesis that classes in inheritance relations are more fault prone. It followed that DIT and NOC metrics [11] could be used to find classes that are likely to have higher fault rates. Some studies chose code smells as predictor of change-proneness. For example, Khomh et al. [12] [13] studied the impact of code smells on software change-proneness and showed that, in their corpus, classes with code smells are more change-prone than others.

Still others concentrated on the effect of programming languages on software quality [14], [15], [16], [17]. For instance, Baishakhi Ray et al. engaged a large scale study of programming languages and code quality in Github. They found that language features, such as static v.s. dynamic typing, strong v.s. weak typing, do have a significant, but modest effect on software quality. Bhattacharya and Neamtiu proposed a novel methodology which controls for development process and developer competence, and evaluates how the choice of programming language affects software quality and developer productivity. Fateman discussed the advantages of Lisp over C and how C itself contributes to the “pervasiveness and subtlety of programming flaws.” The author categorized flaws into various kinds (logical, interface and maintainability) and discussed how the very design of C, e.g., the presence of pointers and weak typing, makes C programs more prone to flaws.

Our study does not claim to compare which one is the best predictor of software quality. On the other hand, we are motivated by the previous work concerning the relation between language features and software quality, and are enthusiastic about how dynamic features may influence change-proneness since they are claimed to have an effect on maintenance.

III. PYTHON DYNAMIC FEATURES

In this section, we first briefly introduce the 18 built-in Python dynamic features we focus on. Then we describe the method to collect them.

A. Dynamic Features Selection and Classification

Although there are multiple kinds of dynamic features in Python language, we choose the 18 famous and most often used and investigated [5], [6], [7] features, as shown in TABLE I, which are thought to be representative and are classified as *Introspection*, *Object Changes*, *Code Generation* and *Library Loading*. For brevity, we refer to the Python Reference Manual [8] and present the definition of each classification stated as follows, instead of a description of the individual constructs.

Introspection is a mechanism to treat modules and functions in memory as objects, getting information about them, and manipulating them.

Object Changes is a category of features that can update or change the state of an object, and that can update, add or remove fields in a way that may depend on the program state.

Code Generation is a category of features that can execute code generated or imported in text format during runtime.

Library Loading is a category of constructs that can load or reload arbitrary libraries at runtime, which allows deferring decisions such as what library should be loaded according to user input or underlying hardware.

TABLE I. PYTHON DYNAMIC FEATURES OF FOUR CATEGORIES

Categories				
<i>Introspection</i>		<i>Object Changes</i>	<i>Code Generation</i>	<i>Library Loading</i>
hasattr	isinstance	setattr	eval	<code>__import__</code>
getattr	issubclass	delattr	exec	Reload
callable	type	del	execfile	
globals	vars			
locals	super			

B. Dynamic Features Collection

Previous works presented two popular methods to collect the use of dynamic features. One is to statically analyze the source code to identify the occurrence of a certain kind of dynamic feature, e.g. CallaúO et al. [1] developed a framework in Pharo² to trace statically the use of dynamic features of Smalltalk. The other is carried out using trace-based dynamic data collection by instrumenting an interpreter to record runtime data [5]. Tracing is able to more precisely describe actual uses of a certain feature than purely static analysis but is sensitive to different paths taken in a program due to input.

In our study, we employ the static collection method instead of the dynamic data collection, because it is difficult to choose representative inputs or interaction strategies that will give acceptable code coverage to figure out all files with or without dynamic features. The specific code analysis and data collection process are supported by a static analysis tool *Understand*³. For each version of a system, we first filter non-Python source files by using the *Python Strict* option in *Understand* to dispose of files unrelated and then build an intermediate database which stores information of entities (function, variable, file, class, attribute et al.), the call graphs among these entities and so forth. After that, we write Perl scripts to invoke *Understand* APIs to mine all program points that use the built-in dynamic features from the database. The algorithm contains three steps:

²<http://www.pharo-project.org>

³<https://scitools.com/>

TABLE II. SUMMARY OF THE CHARACTERISTICS OF THE ANALYZED SYSTEMS

Project	Releases (number)	Duration	Files	LOCs	Description
Boto	2.0-2.28.0 (6)	2011.07-2014.04	217-617	29,246-104,967	interfaces to Amazon Web Services
Bzr	1.2-2.5.0 (9)	2008.02-2012.03	585-830	148,183-263,454	version control system
Django	1.0-1.6 (7)	2008.09-2013.11	956-1872	83,136-165,184	high-level Python Web framework
Matplotlib	0.99.0-1.3.1 (6)	2009.08-2013.10	767-1677	99,934-163,780	library for 2D plotting
Numpy	1.0.4-1.6.2 (8)	2007.12-2012.08	255-398	58,866-119,479	library for mathematics, science, engineering
Scipy	0.7.0-1.13.2 (8)	2009.02-2013.12	419-510	91,479-149,471	library for mathematics, science, engineering
Tornado	1.0.0-3.2.1 (8)	2010.07-2014.05	42-97	10,915-22,095	high-level Python Web framework

1) Firstly, for each function called in a database, the algorithm checks whether it reflects one of the analyzed dynamic features except for *del*, simply by comparing their names. If it matches one, find out the name of the file that uses this function, and thus the number of the matched dynamic features in this file is increased by one.

2) Secondly, for each lexeme in a file recognized by *Understand*, the algorithm checks whether its token is a keyword and its text is equal to *del*. If it is, then record the file name and increase the number of the *del* in this file.

3) Thirdly, for a kind of dynamic feature that does not appear in a file, the algorithm sets the number of that dynamic feature in the file to zero.

4) Finally, the algorithm makes a two-dimensional table stored in .csv format for the subsequent data analysis, which saves all of the file names of a system and the number of each dynamic feature used in every file.

IV. STUDY DEFINITION AND DESIGN

Section four starts with an explanation of how to get change information of each file. Then it presents an introduction of the target systems. After that, it elaborates the research questions and the analysis methods for solving each research question.

A. File Change Information

In the experiment analysis, we need the change information of each file, specifically whether the file is changed or not. To acquire such data, we first write a Perl script to invoke the Linux system command '*diff*' which can be used to compare two arbitrary text files. The execution of the script can generate a formatted difference report textfile that records the position of all the changes and the number of changed lines (added, modified or deleted). Then by writing another script to mine the formatted difference report, we can easily get change data of each file and store them in .csv format likewise. Furthermore, for files that appear in the former version but disappear in the latter version, we identify them as changed files.

B. Data Sets

The context of this study consists of the change history and dynamic features of 7 most famous open-source projects, which have a different size and belong to different domain. For each target system, we regularly choose releases in the interval of 4 to 12 months. Characteristics of the analyzed projects are shown in TABLE II, and the more detailed data are published online¹. On every considered release, we gather the change information

and dynamic features of each file, depending on the methods mentioned earlier.

C. Research Questions

Based on the data collected from the above systems, our study aims to answer 3 research questions.

- RQ1: What is the relation between dynamic features and change-proneness? More specifically, we explore if files with dynamic features are more change-prone than others by testing the null hypothesis: H_{01} : the percentage of files exhibiting at least one change between two releases does not significantly differ between files with dynamic features and other files.
- RQ2: What is the relation between the number of dynamic features in a file and its change-proneness? We analyze whether files with a higher number of dynamic features are more change-prone than others by testing the null hypothesis: H_{02} : the number of dynamic features in change-prone files is not significantly higher than the number of dynamic features in files that do not change.
- RQ3: What is the relation between particular categories of dynamic features and change-proneness? Since, we are also interested to evaluate whether particular categories of dynamic feature contribute more than others to changes by testing the null hypothesis: H_{03} : files with particular categories of dynamic features are not significantly more change-prone than other files.

D. Analysis Methods

To answer RQ1, we test whether the proportion of files undergoing (or not) at least one change significantly varies between files with dynamic features and other files by using Fisher's exact test [18]. This test is appropriate for categorical data that result from classifying objects in two different ways and is used to examine the significance of the association (contingency) between the two kinds of classification. To apply the test, we divide the files of each release into four groups, that is, (1) files undergoing at least one change and with at least one dynamic feature; (2) files undergoing at least one change but with no dynamic feature; (3) files undergoing no change but with at least one dynamic feature; (4) files neither changing nor using dynamic feature. In addition, we compute the odds ratio (*OR*) [18]. The *OR* is the ratio of the odds p of an event occurring in one group, i.e., the odds that files with dynamic features underwent a change (experimental group), to the odds q of it occurring in another group, i.e., the odds that files with no

dynamic features underwent a change (control group), more intuitively: $OR = \frac{p/1-p}{q/1-q}$. An OR greater than 1 indicates that changes are more likely to happen in files with dynamic features, while an OR less than 1 means that changes are more likely to happen in files without dynamic features. If odds ratio equals to 1, the event is equally likely in both samples.

In RQ2, we use the Mann-Whitney test to compare the number of dynamic features in change-prone files with the number of dynamic features in non-change-prone files. The Mann-Whitney test is a non-parametric test that does not require any assumption on the underlying data distributions, and thus is suitable for our experiment. Other than testing the hypothesis, it is of practical interest to estimate the magnitude of the difference of the number of dynamic features in files with and without changes, thus we use the Cohen's d effect size [18]. A d greater than 0 indicates that the number of dynamic features are more in changed files than in not changed files, and less than 0, the contrary. It is worth mentioning that the effect size is often considered small for $0.2 \leq |d| < 0.5$, medium for $0.5 \leq |d| < 0.8$ and large for $|d| \geq 0.8$. For RQ2, we consider the files change or not as the independent variable, and the number of dynamic features in files as the dependent variable.

In RQ3, to relate change-proneness with the presence of particular categories of dynamic features, we use a logistic regression model which is widely used in many studies, e.g., [12], [19], to deal with similar problems. In the logistic regression model, the dependent variable is commonly a dichotomous variable and, thus, only two values $\{0, 1\}$, i.e., in this article changed or not. The multivariate logistic regression model is based on the formula:

$$\pi(X_1, X_2, \dots, X_n) = \frac{e^{\beta_0 + \beta_1 \cdot X_1 + \dots + \beta_n \cdot X_n}}{1 + e^{\beta_0 + \beta_1 \cdot X_1 + \dots + \beta_n \cdot X_n}}$$

where (a) X_t are characteristics describing the modelled phenomenon, in our case, the number of dynamic features of category t a file contains; (b) β_t are the model coefficients; and (c) $0 \leq \pi \leq 1$; the closer the value is to 1, the higher is the likelihood that the file undergoes a change. For each category of dynamic features, we count the number of times that, across the analyzed releases of a target system, the p -values obtained by the logistic regression are significant. If files participating in a specific category of dynamic features are more likely to change in more than 75% of the releases of a target system, then we say that this category of dynamic features has a significant impact on increasing the change-proneness in this system.

V. STUDY RESULTS

In this section, we present the results of our empirical study which are further discussed in section six. More detailed results and raw data are available online¹.

A. RQ1: Dynamic Features and Change-Proneness

TABLE III reports the results of Fisher's exact test and OR values when testing H_{01} . For each target system, it presents the number of all the releases that are analyzed and the number of releases whose p -values of Fisher's test are significant (p -values < 0.05). To be specific, six of seven projects turn out to be significant for more than 75% of their releases, and three projects even prove to be significant for all the releases analyzed. The only outlier is Tornado, five of eight releases turn out to be significant. In summary, although the results sometimes depend on systems analyzed, we can reject H_{01} , i.e., the percentage of files exhibiting at least one change between two releases does significantly differ between files with dynamic features and other files. Regarding the OR s of significant releases, they vary across systems and, within each system, across releases. In 75% of the releases of six systems, the OR s for files with dynamic features to change are two times higher or more than for files without dynamic features and thus odds to change is in general higher for files with dynamic features. In very few releases of some systems, as highlighted, OR s are close to 1, i.e., the odds are even that a file with a dynamic features changes or not.

We therefore conclude that, in most cases, there is a negative relation between dynamic features and change-proneness: a greater proportion of files participating in dynamic features change comparing to other files. Developers should be wary of files with dynamic features, because they are more likely to be the subject of their maintenance effort.

B. RQ2: Number of Dynamic Features and Change-Proneness

TABLE IV presents results of the Mann-Whitney two-tailed test and Cohen's d effect size of the target systems, with the purpose of comparing the number of dynamic features in files that changed or not. More than 75% of the releases of all projects, show significant p -values with relatively small to medium effect sizes, except for Tornado, where only 4 out of 8 releases are significant but with a medium effect size. Moreover, the releases that prove not to have significant p -values confirm the findings from RQ1 regarding the limited relation of dynamic features with change-proneness for these releases. It is worth mentioning that p -value of boto-2.6.0 is significant (p -value=0.02) in RQ2

TABLE III. SUMMARY OF FISHER TEST RESULTS AND OR VALUES FOR EACH TARGET SYSTEM

Project	Number of analyzed releases	Number of significant p-values	Percent of significant p-values	OR					
				Max	Min	Mean	25% quartile	50% quartile	75% quartile
Boto	6	5	83.3%	4.18	1.97	2.83	2.04	2.15	3.96
Bzr	9	8	88.9%	4.77	2.05	3.00	2.44	2.82	3.33
Django	7	7	100%	10.13	1.38	5.88	3.47	4.48	9.53
Matplotlib	6	6	100%	27.07	1.61	8.71	3.78	5.30	13.13
Numpy	8	8	100%	4.77	2.19	3.47	2.71	3.54	4.31
Scipy	8	6	75%	4.04	1.50	2.87	1.69	3.30	3.91
Tornado	8	5	62.5%	8.70	3.94	5.96	4.08	6.41	7.61
Sum	52	45	86.5%	-	-	-	-	-	-

TABLE IV. SUMMARY OF MANN-WHITNEY RESULTS AND COHEN'S D FOR EACH TARGET SYSTEM

Project	Number of analyzed releases	Number of significant p-values	Percent of significant p-values	Cohen's d					
				Max	Min	Mean	25% quartile	50% quartile	75% quartile
Boto	6	6	100%	0.60	0.16	0.39	0.27	0.38	0.56
Bzr	9	8	88.9%	0.45	-0.01	0.34	0.30	0.38	0.44
Django	7	7	100%	0.61	0.07	0.35	0.28	0.36	0.41
Matplotlib	6	6	100%	0.82	0.12	0.45	0.21	0.40	0.73
Numpy	8	8	100%	0.55	0.05	0.38	0.29	0.43	0.51
Scipy	8	6	75%	0.55	0.33	0.44	0.38	0.45	0.50
Tornado	8	4	50%	0.75	0.54	0.64	0.56	0.64	0.73
Sum	52	45	86.5%	-	-	-	-	-	-

but not significant (p -value=0.14) in RQ1, yet we consider it a tolerable abnormal phenomena that does not affect the whole results. In summary, the results of most releases support that change-prone files are those with a higher number of dynamic features and thus we can reject H_{02} .

C. RQ3: Categories of Dynamic Features and Change-Proneness

TABLE V summarizes the results of the logistic regression for the correlations between change-proneness and the different categories of dynamic features. In particular, the table presents the number of analysed releases for which each categories of dynamic features is significant in the logistic regression model. Boldface indicates significant p-values for at least 75% of the releases in each system. Following our analysis method of RQ3 in section four, it is noticed that *Introspection* is shown to be significantly correlated to change-proneness in 5 target systems, and that *Library Loading* only has impact on Numpy project. However, for Boto and Tornado, there are not enough releases to support the relation between any category of dynamic features and change-proneness. Therefore, we can partly reject H_{03} for *Introspection* and *Library Loading* depending on the results observed. On the whole, although only 5 of 7 analyzed systems reject H_{03} , we can conclude that there are categories of dynamic features which are more related to others to change-proneness in most cases and that the relation between particular categories of dynamic features and change-proneness cannot be completely ignored. What is more, the *Introspection* category deserves extra attention for it turns out to be more related to change-proneness than others.

TABLE V. NUMBER OF RELEASES WHERE EACH CATEGORY OF DYNAMIC FEATURES SIGNIFICANTLY CORRELATES WITH CHANGE-PRONENESS.

Project	Number of analyzed releases	Proneness to Change of each category of Dynamic features			
		<i>Introspection</i>	<i>Object Changes</i>	<i>Code Generation</i>	<i>Library Loading</i>
Boto	6	3	2	-	-
Bzr	9	7	4	1	-
Django	7	6	2	1	1
Matplotlib	6	5	3	2	1
Numpy	8	6	-	2	6
Scipy	8	6	-	2	-
Tornado	8	3	1	-	-

VI. DISCUSSION

We now discuss the implications of the results reported in section five, along with threats to validity.

A. Discussions and Implications

In this study, we investigate the impact of 18 built-in Python dynamic features on file change-proneness. As analyzed in section five, the results show that files with dynamic features (and, in particular, those with a higher number of dynamic features) are significantly more change-prone than others in most releases of the analyzed systems, except for Tornado. And dynamic features of *Introspection* are more related to file change-proneness than the other three categories. Based on these results, we can get some useful implications for both research and practice.

For the research community, this work is the first one to focus on the relation between dynamic features and maintenance. The negative relation between dynamic features and change-proneness promotes further investigations to be conducted on the relation between dynamic features and other maintenance related factors, such as fault-proneness. In sum, our study inspires researchers to turn their attention from how and why to use dynamic features to the effect that these features have on maintenance. Additionally, we suggest that more work should be focused on the category of dynamic features that affect change-proneness most, in this work, the *Introspection* category, and on how and why this kind of feature can be constructed, in order to improve the quality of software and help us better understand dynamic features as well.

For practice, we suggest that developers should be cautious when using dynamic features, especially the *Introspection*, because the presence of these features may lead to the maintenance effort and cost. As for quality assurance personnel, they need to pay extra attention to files with more dynamic features, since these files may contribute to more maintenance problems.

In addition to the foregoing, it is noticed that Tornado does not exhibit an overwhelming significant relation (percent of significant p -values $\geq 75\%$) of all the releases even if in one of the three RQs. We deduce the reason for this fact lies in the minor number of files of each release ranging from 42 to 97, while file number of the other systems varies from hundreds to thousands.

B. Threats to Validity

Internal threats in this work mainly concern whether the hypothesis testing methods are properly used. Although in practice the Fisher's exact test is often employed when sample sizes are small, it is also valid for all the sample sizes. Also, we

choose the non-parametric tests that do not require making assumption about the data set distribution. To build the logistic regression model, it is important to discard the independent variables that are highly correlated to each other. We eliminate such a threat by calculating the Spearman rank correlation coefficient between any two different categories of dynamic features. As expected, the results³ show that no two categories of dynamic features are highly correlated (Spearman rank correlation coefficient is higher than 0.8), and thus it is no need to exclude any of the independent variables in our experiment.

Threats to external validity concern the possibility to generalize our findings. Although we have tried our best to limit such a threat and make the results general by choosing 7 open-source systems of 5 different problem domains, as shown in TABLE I, and by covering most of the built-in Python dynamic features that are representative in each of the categories, yet the generalization still requires further case studies including a large number of Python systems from various domains and more dynamic features as well. Besides, since covering all historical versions for one project is a hard work, we select them regularly by an interval of 4 to 12 months, which is a reasonable way.

Construct validity threats concern the relation between theory and observation. In our context, they are mainly due to errors introduced in measurements. In this work, the count of changes occurred to files is based on comparing the difference of files with the same name but from two versions. We are just interested to check whether a file changes or not, rather than quantifying the amount of change, which is however possible based on rules in [20] and could be investigated in the future work. In our detection algorithm, we ignore dynamic features appearing in annotated codes. But we consider it does not influence our results, for these circumstances are rare and are often used for illustration purpose not for realizing functions.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we explore how the use of dynamic features affects file change-proneness. The whole study is undertaken by choosing 18 most often used and studied Python dynamic features [5], [6], [7] and 7 famous open-source Python systems from Github and SourceForge online repositories. We find that files with dynamic features are significantly more likely to be the subject of changes, than other files. We also show that dynamic features of *Introspection* are more likely to be of concern during evolution. This exploratory study supports, within the limits of the threats to its validity, the conjecture in the literature that dynamic features may have a negative impact on software evolution. Depending on the results observed, we suggest practitioners that they should be cautious of treating systems with a high prevalence of dynamic features during development and maintenance, because those systems are likely to be more change-prone: therefore, the cost-of-ownership of such systems will be higher than for other systems. Additionally, we call on researchers to pay more attention to dynamic features of other languages concerning their impacts on software quality and on the root causes of their negative impact, on the basis of our work.

In the future work, we will replicate this study on more systems and with more dynamic features considered to validate the above-mentioned findings. Further, we are interested to

relate dynamic features to other phenomena such as the fault-proneness.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (61472175, 61170071, 61472178), and the National Natural Science Foundation of Jiangsu Province (BK20130014). I express my sincere gratitude to all the teachers and students who make contributions to this work.

REFERENCES

- [1] Callaú O, Robbes R, Tanter É, Röhlsberger D. How (and why) developers use the dynamic features of programming languages: the case of Smalltalk. *Empirical Software Engineering* 18.6 (2013): 1156-1194.
- [2] An, J. H. D., Chaudhuri, A., Foster, J. S., & Hicks, M. (2011). Dynamic inference of static types for ruby (Vol. 46, No. 1, pp. 459-472). *ACM*.
- [3] Richards, G., Lebesne, S., Burg, B., & Vitek, J. (2010, June). An analysis of the dynamic behavior of JavaScript programs. In *ACM Sigplan Notices* (Vol. 45, No. 6, pp. 1-12).
- [4] Richards, G., Hammer, C., Burg, B., & Vitek, J. (2011). The eval that men do. In *ECOOP 2011—Object-Oriented Programming* (pp. 52-78).
- [5] Åkerblom, B., Stendahl, J., Tumlin, M., & Wrigstad, T. (2014, May). Tracing dynamic features in python programs. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (pp. 292-295).
- [6] Tratt, L. (2009). Dynamically typed languages. *Advances in Computers*, 77, 149-184.
- [7] Holkner, A., & Harland, J. (2009, January). Evaluating the dynamic behaviour of Python applications. In *Proceedings of the Thirty-Second Australasian Conference on Computer Science-Volume 91* (pp. 19-28).
- [8] G. van Rossum and F.L.Drake, "PYTHON 2.6 Reference Manual", CreateSpace, Paramount, CA, 2009.
- [9] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *TSE*, 22(10):751-761, 1996.
- [10] M. Cartwright and M. Shepperd. An empirical investigation of an object-oriented software system. *TSE*, 26(8):786-796, August 2000.
- [11] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6), 476-493.
- [12] Khomh, F., Di Penta, M., & Gueheneuc, Y. (2009, October). An exploratory study of the impact of code smells on software change-proneness. In *Reverse Engineering, 2009. WCRE'09. 16th Working Conference on* (pp. 75-84). IEEE.
- [13] Khomh, F., Di Penta, M., Guéhenec, Y. G., & Antoniol, G. (2012). An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Empirical Software Engineering*, 17(3), 243-275.
- [14] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar T Devanbu. A Large Scale Study of Programming Languages and Code Quality in Github. *FSE '14*, 16-22, 2014.
- [15] S. Hanenberg. An experiment about static and dynamic type systems: Doubts about the positive impact of static type systems on development time. *OOPSLA '10*, 22-35, 2010.
- [16] Fateman, R. (2002). Software fault prevention by language choice: Why C is not my favorite language. *Advances in Computers*, 56, 167-188.
- [17] Pamela Bhattacharya and Iulian Neamtiu. Assessing Programming Language Impact on Development and Maintenance: A Study on C and C++. *ICSE '11*, 21-28, 2011.
- [18] D. Sheskin. Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition). Chapman & All, 2007.
- [19] Hosmer Jr, D. W., & Lemeshow, S. (2004). *Applied logistic regression*. John Wiley & Sons.
- [20] Yuming Zhou, Hareton Leung and Baowen Xu. Examining the potentially confounding effect of class size on the associations between object-oriented metrics and Change-Proneness. *TSE*, 607-623, 2009.

Evaluating Software Engineers' Acceptance of a Technique and Tool for Web Usability Inspection

Luis Rivero¹, Auri Vincenzi^{2,3}, José Carlos Maldonado³ and Tayana Conte¹

¹USES Research Group, Instituto de Computação, Universidade Federal do Amazonas, Manaus – Brazil

²Instituto de Informática, Universidade Federal de Goiás, Goiânia – Brazil

³Departamento de Ciência da Computação, Universidade de São Paulo (USP), São Carlos – Brazil
{luisrivero, tayana}@icomp.ufam.edu.br; auri@inf.ufg.br; jcmaldon@icmc.usp.br

Abstract— Usability is related to software quality, improving its ability to be understood, operated and attractive to users. We proposed the Design Usability Evaluation (DUE) technologies to allow identifying usability problems earlier in the development of Web applications, through the inspection of mockups. While we found that the DUE technique and tool were effective and efficient in the identification of usability problems, we saw the need to investigate their acceptance in practitioners' work environment. This paper reports the results from a study evaluating the acceptance of the DUE technologies from the point of view of software engineers. We asked questions based on the indicators from the Technology Acceptance Model and identified that a majority of the software engineers who participated in the study: (a) found the DUE technologies useful and easy to use for supporting the usability inspection process; and (b) would regularly use the DUE technologies for future inspections in their job. Nevertheless, the practitioners indicated that the technique should be refined in order to reduce the ambiguity and repetition of some of its items, while the tool should become more intuitive.

Keywords- *Web usability; software quality; inspection technique; inspection tool; software testing tool; empirical study; technology acceptance*

I. INTRODUCTION

A Web Application is a software system based on technologies and standards of the World Wide Web Consortium (W3C¹) that provides Web specific resources such as content and services through a user interface, the Web browser [1]. Due to their importance for presenting products and services to customers, Web applications need to be usable so that they can be effective, efficient and satisfying to users [1]. In that context, usability subsumes aspects such as learnability, operability, aesthetics, and others that affect the quality of the developed applications [2]. Therefore, usability plays a central role in their acceptance and adoption [3].

Usability inspection is one of the ways for identifying usability problems, in which inspectors check the conformity of software artifacts against a set of usability standards [1]. However, although the number of usability inspection methods for evaluating Web applications has increased, only a short number of these methods can be applied earlier in development [4]. Methods applied later in the development process, or when the application is released, can increase the cost of correcting

the identified problems since the source code of the application will have already been written [5]. Also, the difficulty in identifying usability problems increases if the inspectors are not guided through the evaluation process or if they do not have tool support for reducing cognitive effort while performing an inspection [4].

The positive reports on the use of mockups (sketches of how an application would look like after its development) to support several early software engineering activities [6] motivated us to develop a set of technologies for the usability inspection of mockups of Web applications [4]. These technologies, called Design Usability Evaluation (DUE), provide a technique and tool. While the technique provides a set of verification items that guide inspectors through the evaluation process, the tool facilitates its application by simulating interaction among the evaluated mockups and allowing pointing usability problems and generating reports.

In our previous work, we conducted empirical studies, showing indicators of the feasibility of the DUE technologies in the usability inspection of mockups in terms of effectiveness and efficiency in different conditions and when compared with other techniques [7][8]. However, these studies were carried out in academic environments or did not focus on the aspects that needed to be improved to enhance the acceptance of DUE technologies in a real usage scenario. A good understanding of real software engineers' attitude towards the DUE technologies is expected to help us decide whether and how the technologies should be tailored to improve the results of the usability inspection of the mockups of Web applications.

According to Shull et al. [9], studies in a particular development lifecycle can help evaluating if new proposed software engineering technologies are compatible with software engineers' work environment. The results from these studies can reveal issues that did not arise during feasibility studies, allowing fine-tuning or tailoring of the technology to meet the needs of the software industry. This type of studies is essential for the industry in order to decide whether they will adopt or reject a specific technology. Therefore, following our evaluation of the feasibility of the DUE technologies in terms of effectiveness and efficiency [7], this paper presents a study in a real lifecycle. In this study, software engineers with experience in software verification and validation tried the DUE technologies and reported their perceived usefulness and perceived ease of use towards them. The goal of this paper is to

¹ <http://www.w3.org/>

report on their perception of the DUE technologies and their degree of acceptance. Additionally, we gathered data on what would make practitioners adopt or reject the DUE technologies.

II. BACKGROUND AND MOTIVATION

According to Fernandez et al. [10], in order to assist the identification of usability problems in Web applications, new research has been performed in the field of usability evaluations. Such evaluations can range from [4]: (a) User Testing, in which users perform tasks so that an observational team can identify communication gaps and usability problems regarding the user interface; and (b) Usability Inspection Methods (UIMs), in which inspectors verify the conformity of software artifacts against a set of usability standards. The main advantage of applying inspection methods is that they require fewer resources to be applied. Since UIMs do not require special equipment or laboratories to be executed, they can lower the costs of the identification of usability problems [1].

In our previous work [4], we carried out an analysis over the review by Fernandez et al. [10], gathering data on UIMs for Web applications. Among the analyzed methods, Paganelli and Paterno [11] proposed a UIM that compares the way in which a Web system is expected to be used and the way in which it is really used, to identify usability problems. Also, Allen et al. [12] developed the Paper-Based Heuristic Evaluation, an inspection method evaluating mockups of medical Web applications in terms of consistency, minimalism, match, memory and language. Finally, Molina and Toval [13] proposed a method that provides a total of 50 metrics in order to identify usability problems from a meta-model formed by merging the navigational and requirements models.

Although the above methods provide means for identifying usability problems in Web applications, there is still room for improvement. For instance, methods such as the one proposed by Paganelli and Paterno [11] require that the source code of the application is available so users can experience it, which increases the cost of correcting the identified problems [5]. In fact, our analysis [4] showed that only around 15% of the identified usability evaluation methods could be applied at earlier stages of the development process. Moreover, from the methods that can be employed earlier in development (e.g. the Paper-Based Heuristic Evaluation), most of them do not provide guidance for software engineers applying them and/or do not provide tools to support inspectors in the evaluation of Web applications [4]. It is necessary to develop technologies (i.e. methods and tools) that address these issues in order to enhance the performance of software engineers in the evaluation of Web applications. The above needs have been considered in the proposal of the DUE technologies.

III. THE DUE TECHNOLOGIES

We proposed the Design Usability Evaluation (DUE) technologies in order to meet the needs of inspection methods in the field of early usability evaluation of Web applications [4]. In this sense, the DUE technologies are a technique and a tool to guide software engineers in the identification process of usability problems in mockups of Web applications.

To guide inspectors in the identification of usability problems, the DUE technique suggests dividing mockups into

Web page zones, which are pieces of a Web page that contain specific components to perform certain functionalities [4]. Examples of Web page zones are: the navigation zone, where the user can find means to go from one part of the application to another; the system state zone, where the user can find information of his/her location in the application, how (s)he got there and the available options in it; and others. Based on these zones, the DUE technique provides a set of verification items to check whether a usability problem can occur. For instance, Table I shows some of the verification items of the DUE technique for the data entry zone, while Fig. 1 shows a mockup in which these items have been violated. When an inspector verifies that there is a nonconformity, (s)he marks the item within the mockup, identifying the problem. As an example, problem P01 in Table I shows that in the data entry zone, the Web application does not provide hints for filling the fields, which can cause difficulties in inputting data (see Fig. 1 error A). Also, problem P02 in Table I shows that the fields that are mandatory are not highlighted, which can cause users to forget to provide relevant information (see Fig. 1 error B). Interested readers can find a complete description of the Web page zones and verification items from the DUE technique, and more examples of usability problems in our previous work [4].

TABLE I. A WEB PAGE ZONE AND SOME OF ITS VERIFICATION ITEMS.

Data Entry Zone: This zone is responsible for providing the user with means of entering data into the application in order to allow the user to perform operations. Later, the user will click in a “submit” like button that will activate a function based on the entered data.	
ID	Usability Verification Item
P01	The interface indicates the correct format for a determined data entrance (e.g. a “Date” entry field could have the next hint: “mm/dd/yy”).
P02	The interface indicates which data must be mandatory filled (e.g. mandatory input data is indicated with a “*” or a “mandatory” next to the field).

The initial evaluation of the DUE technique [7] showed that applying the technique on its own was tiring for inspectors, as they were forced to simulate the interaction between the user and the mockups. Therefore, we proposed the DUE tool to facilitate the application process of the DUE technique. To simulate navigation, the DUE tool allows inspectors to click on previously added links that, when activated, show the mockups in a sequence, to resemble a real application. Also, the DUE tool embeds the verification items from the DUE technique and shows them to the inspectors next to the set of evaluated mockups. This way, inspectors can request further information and details on the verification items while identifying usability problems. Fig. 1 shows the DUE tool when employed in the evaluation of a mockup. Using the tool, inspectors can point errors and notes (see area of the screen indicated by 1), view the verification items of the technique and simulate interaction (see areas of the screen indicated by 2 and 3 respectively).

Although there are other tools for creating mockups and simulating interaction (e.g. Mockingbird², Balsamiq Mockups³ and others), these tools do not provide specific support for the usability inspection of the developed mockups. It is possible for

² <https://gomockingbird.com/>

³ <http://www.balsamiq.com/>

inspectors to use these tools to simulate interaction, while identifying usability problems. However, it would be difficult to point the exact location of the encountered problems within the mockups while navigating through the application using these tools. In that context, the DUE tool allows inspectors to save the data on the inspection without using a spreadsheet, which would facilitate to pause and resume the inspection process whenever it is necessary. These are the main advantages of the DUE tool when compared to other tools.

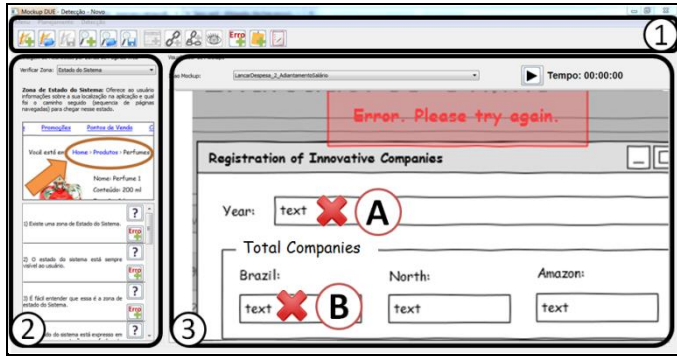


Figure 1. The DUE tool in its evaluation mode: (1) functionality bar, (2) verification items from the DUE technique, and (3) mockup being evaluated.

In order to complete an inspection using the DUE technologies, the inspector must evaluate all the verification items for all the zones present within the evaluated mockups. Then, (s)he can generate an automatic report containing information on the inspector and the identified problems. Later, such report will be discussed by the development team in order to verify which corrections are necessary and their priority.

IV. EMPIRICAL EVALUATION

In this study, we aimed at: (a) assessing the acceptance of the DUE technologies by software engineers; and (b) identifying constrains and improvement opportunities for adopting these technologies in the software industry. To gather data for evaluating the acceptance of the DUE technologies, we applied a questionnaire based on the indicators from the Technology Acceptance Model (TAM). TAM [14] aims at assessing users' beliefs about the usefulness and ease of use of a technology that is expected to support their work. According to Davis [14], the reason for focusing on those indicators is that they are strongly correlated to user acceptance of a given technology.

The empirical study to assess the DUE technologies was conducted during a two-week professional training on software verification and validation. The goal of the course was to teach software engineering practitioners about new techniques and tools for guaranteeing quality in the software development lifecycle. One of the topics from the course was usability evaluations and our research team was asked to provide training on technologies for evaluating the usability of different software applications and their suitability in different stages of the development process. Thus, a training regarding the DUE technologies was prepared as part of the course.

Table II shows the questionnaire we applied for evaluating perceived usefulness and perceived ease of use on the DUE technologies. We based our questionnaire on the one by

Laitenberger and Dreyer [15]. However, we selected only part of the items that could provide information on what could be improved in the development of the DUE technologies. In order to apply the questionnaire, we:

- Replaced the “technology” investigated in the questionnaire with the terms “DUE technique” or “DUE tool” according to the technology we were evaluating.
- Replaced the process investigated in the questionnaire with “usability inspection” with a focus on Web mockups.
- Employed a four-point scale asking for the degree of agreement with the statements from the point of view of software engineers: (1) Strongly Disagree, (2) Partially Disagree, (3) Partially Agree and (4) Strongly Agree. We did not use an intermediate level so the software engineers would provide information regarding the side to which they were inclined (either positive or negative) [15].
- For each of the statements within the questionnaire, we included open questions, asking for the reason why a subject chose a specific answer. This was done in order to better understand the features that made the DUE technologies useful (or useless), easy (or difficult) to use and suitable (unsuitable) for a software engineer’s work.

At all, 20 software engineers from 5 different software companies (at Manaus-Brazil) were enrolled in the training and agreed to participate in the study. These software engineers had a strong technical background (knowledge in the planning, creation and documentation of test cases), and varying degrees of work experience in the testing of software applications (ranging from 2 to 10 years of experience – median 5 years).

The study took place in two days from the two-weeks training. Each day, the subjects entered a lab room where they had lectures and carried out real evaluations for a period of 4 hours with a 30 minutes break. During the first half of the training of the first day, the subjects received training in the application of different usability evaluation techniques (e.g. user testing [1], the heuristic evaluation [16]). After that, the subjects performed evaluations using these techniques over real applications under development. Then, during the second half of the course of the first day, the subjects were trained on the DUE technique, applied on its own without tool support, for inspecting the usability of the mockups of a Web application. Next, they performed the evaluation of a set of mockups and filled in the questionnaire statements regarding their acceptance of the DUE technique. During the second day, the subjects had training on the DUE tool. However, this time, they had to carry out an inspection over a real application under development comprising over 10 mockups. This was done in order to resemble a real evaluation scenario in industry and let the subjects experience the navigation functionalities from the DUE tool. Finally, all subjects filled in the questionnaire with statements regarding their acceptance of the DUE tool. We highlight that we did not compare the DUE technologies with other usability evaluation approaches as this study focused on the acceptance of the DUE technologies by software engineers. Studies comparing the DUE technologies to other approaches can be found in our previous work [7].

TABLE II. QUESTIONNAIRE STATEMENTS ON: PERCEIVED USEFULNESS, EASE OF USE AND FUTURE USE.

Statements regarding "Perceived Usefulness" (U):	
U1	Using the "technology" in my job would <i>improve my effectiveness</i> in a usability inspection of the mockups of a Web application.
U2	Using the "technology" in my job, I would be able to carry out a usability inspection of the mockups of a Web application <i>more quickly</i> .
U3	Using the "technology" in my job would <i>improve my performance</i> in a usability inspection of the mockups of a Web application.
U4	I would find the "technology" <i>useful</i> to carry out a usability inspection of the mockups of a Web application.
Statements regarding perceived "Ease of Use" (EoU):	
EoU1	<i>Learning to operate</i> the "technology" to carry out a usability inspection of the mockups of a Web application <i>would be easy</i> for me.
EoU2	I would find it easy to get the "technology" <i>to do what I want it to do</i> to carry out a usability inspection of the mockups of a Web application.
EoU3	It would be <i>easy to become skillful in using</i> a usability inspection technique/tool like the "Technology".
EoU4	I would find a usability inspection technique/tool like the "Technology" <i>easy to use</i> .
Statements regarding "Self-Predicted Future Use" (FU):	
FU1	Assuming a usability inspection technique/tool like the "Technology" would be available on my job, <i>I predict that I will use it on a regular basis in the future</i> .

V. DATA ANALYSIS AND RESULTS

Usefulness and Ease of Use are important measures for technology acceptance. We used the questionnaire to gather software engineers' opinion about their acceptance of the DUE technologies. Table III shows the descriptive statistics for the Usefulness statements (U1 to U4), Ease of Use statements (EoU1 to EoU4) and Self-Predicted Future Use. We have analyzed the results verifying the mean and standard deviation of the scores as in the examples by Laitenberger and Dreyer [15] and Babar et al. [17]. An average response between 3 (Partially Agree) and 4 (Strongly Agree) seems overall a positive result. The overall score for perceived usefulness and perceived ease of use has been calculated by summing the individual scores of their respective items, thus the maximum score is 16. Despite the cautiously positive results, some subjects were not convinced about the usefulness and ease of use of the DUE technologies. To better understand the reasons that made the subjects answer positively or negatively, we have analyzed the answers to the open questions.

Reasons that made software engineers believe that the DUE technique was useful were regarding the guidance and standards that were provided. For instance, one of the subjects indicated that since the technique focused on specific parts of the application and its attributes, it was easier to concentrate and identify the usability problems (see quote from Inspector I10). Furthermore, the verification items and their detailed description according to the zones that were being evaluated made software engineers believe that they would be able to find more problems (see quote from Inspector I04).

"(...) it allows me to focus on different areas and inspect them independently." – Inspector I10.

"I believe it is effective as it supports identifying the defects through its well described items." – Inspector I04.

Overall, 5 inspectors disagreed with at least one of the items on the usefulness of the DUE technique. The main reasons for their disagreement was regarding the time it would take to carry out the inspection due to the large number of verification items (see quote from Inspector I08); and the overlapping between some of the items, which could confuse inspectors when looking for usability problems (see quote from Inspector I17).

"It can be tiring and take a long time depending on how many of the verification items you check." – Inspector I08.

"The number of items and, in some cases, their ambiguity makes it diminish my performance and it takes time." – Inspector I17.

TABLE III. MEAN AND STD. DEV. FOR USEFULNESS, EASE OF USE AND SELF-PREDICTED FUTURE USE (FOUR-POINT SCALE: 1 TO 4).

Item	DUE Technique		DUE Tool	
	Mean	Std. Dev.	Mean	Std. Dev.
U1 - Effectiveness	3,70	0,46	3,58	0,49
U2 - Quick	3,15	0,79	3,50	0,65
U3 - Performance	3,60	0,49	3,50	0,50
U4 - Useful	3,40	0,73	3,08	0,86
Total Usefulness	13,85	1,68	13,67	1,37
EoU1 - Easy to learn	3,70	0,46	3,50	0,50
EoU2 - Controllable	3,50	0,67	3,33	0,62
EoU3 - Skillful	3,60	0,58	3,58	0,49
EoU4 - Easy to use	3,30	0,84	3,42	0,64
Total Ease of Use	14,10	2,14	13,83	2,07
Self-Predicted Future Use	3,50	0,50	3,50	0,65

Regarding the ease of use of the technique, the software engineers indicated that it was easy to identify usability problems as the technique pointed, for the different parts of the application, what an application should provide to be usable (see quote from Inspector I02). Furthermore, the software engineers indicated that the organization of the technique made it easier to learn and follow the inspection process (see quote from Inspector I10).

"It makes it easier since the zones and items make it clear what an application should provide, and what it actually has/lacks." – Inspector I02.

"Yes, using the zones makes it easier to follow the process and identify specific problems in which we would not focus in other circumstances." – Inspector I10.

Despite the positive feedback on ease of use, around 4 software engineers disagreed with at least one of the statements from the questionnaire regarding the DUE technique. Again, the main problem was the overlap between some of the verification items (see quote from Inspector I13). Moreover, other inspectors indicated that initially, the zones were not that intuitive, but as there were examples, one could learn how to use the technique (see quote from Inspector I18). We highlight that some inspectors suggested developing a tool support for the DUE technique (see quote from Inspector I14). While using the DUE technique, the software engineers participating in the study did not know that a tool was available. Thus, it can be an indicator supporting our results in our literature review and previous studies [4][7], which suggest that a tool is important for facilitating the use of UIMs.

“It was not that easy to use as some of the items are ambiguous or overlap, which makes it confusing and take more time” – Inspector I13.

“Initially, I had some difficulty in understanding the zones and their items. However, the examples made me overcome that problem, and I was able to apply it.” – Inspector I18.

“I believe that the technique is suitable if the evaluation is short. However, in bigger applications, it would be better to have a tool to facilitate its use.” – Inspector I14.

Regarding the DUE tool, and its use on the evaluation of the mockups of a real Web application under development, most software engineers provided positive feedback. When asked about the reasons that made the tool useful and easy to use, the inspectors indicated that the tool was useful as it made the inspection process more agile and quick (see quote from Inspector I13). Furthermore, the tool was perceived as intuitive and easy to use as the provided functionalities were easy to understand (see quote from Inspector I10).

“I believe it is a great tool, it makes the inspection process more agile and it makes it easier. It is an adequate and useful tool for the inspection.” – Inspector I13.

“In my opinion, the tool was useful and it was easy to understand the provided options. Also, the way in which the errors are documented helped me. It is very intuitive.” – Inspector I10.

At least 4 software engineers disagreed with one or more statements regarding the usefulness and ease of use of the DUE tool. These inspectors indicated that since the technique had many zones and verification items to be checked, finding them in the tool was also difficult and make using the tool inefficient (see quote from Inspector I08). Also, the appearance of the tool and the way it presented some feedback to the inspectors were not adequate in certain situations. For instance, Inspector I19 pointed out that the way in which the tool pointed the defects made it hard to visualize an application with many defects. In Fig.1 we can see that for each identified problem, the tool adds an “X” mark next to the problem (the inspector can relocate the X over the problem to make that problem easier to find in the report). When a mockup has many problems, as the number of marks increases, it turns difficult to view the mockup. Finally, the inspectors indicated that the inspection report should be reduced (see quote from Inspector I12), as it shows all the evaluated mockup, even if usability problems were not identified on them, thus wasting the time of the development team, when reviewing the reports.

“Since it presents all the zones and items from the technique, it is also tiring. Furthermore, it is difficult to identify previous problems that were added when identifying a usability problem.” – Inspector I08.

“When we report a problem, the tool adds an ‘X’ to point it on the mockup. However, as the problems were being reported I was forced to relocate them so they would not make it difficult to navigate and view the mockups.” – Inspector I19.

“I think that the report contains too much information. It could show the mockups in which problems have been found

instead of showing all of them and wasting time.” – Inspector I12.

When asked if they would employ the DUE technologies in their work environment, the majority of the subjects (strongly or partially) agreed that they would use it. However, only Inspector I19 disagreed with adopting the DUE tool in his/her job. The reason for this answer was that (s)he did not like the tool because of its design. Other inspectors indicated that the tool could be improved by grouping its functionalities (and buttons) according to their frequency of use, and providing shortcuts to make it faster to use. Also, they indicated the need for facilitating the navigation among the mockups and, perhaps, allowing importing mockups from other tools, instead of creating them elsewhere and mapping them into the tool. Finally, they indicated that in the first use, the tool should provide a quick introduction, so inspectors can be more familiar with its functionalities before starting the inspection. Regarding the DUE technique, the software engineers suggested creating generic items for those that were repeated in the zones. Also, they suggested making the ambiguous verification items more clear, by adding further information and hints on what a usable interface should provide.

VI. LIMITATIONS OF THE STUDY

Regarding the subjects’ need for training, it would have been better if there was no need for it. However, the short training time allows the DUE technologies to be applied by software engineers with low experience in usability evaluations. In that context, the moderator and training could have caused an effect in the software engineers’ perception of usefulness and ease of use. Nevertheless, the moderator did not highlight the (dis)advantages of the DUE technologies. Instead, he explained their application process and provided equivalent examples for all methods described in the training. Furthermore, when filling out the questionnaire, the moderator highlighted that the goal of the study was to identify improvement opportunities in the DUE technologies, encouraging the software engineers to be as honest as possible. Finally, besides the DUE technologies, the software engineers applied different usability evaluation techniques to guarantee that they could have a baseline to compare them. However, as the duration of the study depended on the duration of the training, we only gathered data on the acceptance of the DUE technologies.

Regarding the generalization of our findings, the representativeness of the inspected mockups can be a limitation. Although these mockups might not be representative of all types of applications [1] and inspectors may have different results evaluating other applications, these mockups were produced for a real system under development, resembling a real industrial usability evaluation scenario. Therefore, the results from this study must be considered indicators and further studies evaluating different types of applications should be executed. Also, since the number of subjects is low, the data extracted from this study can only be considered indicators and not conclusive. Nonetheless, it might not be possible to get sufficient size of data sets. Therefore, even with a small sample used, the results from this study are

good indicators for explaining the reasons why users would accept or reject the DUE technologies.

A final limitation could be the instrument and measures applied in this study for assessing technology acceptance. However, we believe that applying questionnaires was more suitable than applying interviews due to time constraints. Furthermore, by evaluating perceived usefulness and perceived ease of use, we intended to have an idea of users' acceptance of the DUE technologies and identify issues that should be corrected to meet the needs of the software industry. Finally, the questions we asked to the software engineers were based on questionnaires applied in other researches [15][17] which have been previously validated.

VII. CONCLUSIONS AND FUTURE WORK

We developed a set of usability inspection technologies for the evaluation of mockups of Web applications earlier in their development process. In this paper, we have studied the user acceptance of these technologies for carrying out usability evaluations. We used a questionnaire evaluating indicators based on the TAM model, for gaining understanding of the subjects' attitude towards the DUE technologies for inspecting the usability of mockups of Web applications. In that context, we found out that:

1. A majority of the subjects found the DUE technique and tool quite useful and easy to use for supporting the usability evaluation of mockups of Web applications.

2. Most of the software engineers who participated in the study would adopt the DUE technologies in their job.

3. The practitioners who disagreed with the statements from the questionnaire in terms of usefulness and ease of use indicated that to improve their performance, the DUE technique should reduce or combine some of its verification items and make them less ambiguous.

4. It is necessary to improve the design of the DUE tool to make it easier in its first usage experience. Also, the way in which problems are pointed should be improved so the visualization and navigation among the mockups are not affected.

As we had already evaluated the effectiveness and efficiency indicators of the DUE technologies in different contexts [7], this paper focused on the evaluation of their acceptance by software engineers. However, we still need to carry out further studies verifying to what extent previous knowledge on usability evaluations and previous practical experience affect the acceptance and performance of practitioners when applying the DUE technologies. Thus, as future work, we intend to replicate this study, but increasing the number of subjects, and analyzing their actual effectiveness and efficiency according to their experience. Also, in this new study, it is necessary to implement the changes suggested by the software engineers in order to improve the usefulness and ease of use of the DUE technologies and their adoption in the software industry. Furthermore, although we analyzed the answers to the open questions, we still need to carry out further

qualitative analyses with other methods to better investigate the aspects that need to be improved to enhance their adoption.

ACKNOWLEDGMENTS

We thank CNPq for the scholarship granted to the first author of this paper and for its financial support through process n° 460627/2014-7. Also, we thank the financial support granted by FAPEAM through processes n°: 01135/2011; 062.00146/2012; 062.00600/2014; 062.00578/2014; and PAPE 004/2015. Finally, we thank the financial support granted by FAPESP through processes n° 2014/15514-2.

REFERENCES

- [1] G. Kappel, B. Proll, S. Reich, W. Retschitzegger, "An Introduction to Web Engineering", In: G. Kappel, B. Proll, S. Reich, W. Retschitzegger, "Web Engineering: The Discipline of Systematic Development of Web Applications", John Wiley & Sons, pp. 1-17, 2006.
- [2] International Organization for Standardization, ISO/IEC 25010, "Systems and software engineering – SQuARE - Software product Quality Requirements and Evaluation", 2011.
- [3] L. Olsina, G. Covella G. Rossi, "Web Quality", In: E. Mendes, N. Mosley, Web Engineering, Springer, pp. 109-142, 2006.
- [4] L. Rivero, R. Barreto, T. Conte, "Characterizing Usability Inspection Methods through the Analysis of a Systematic Mapping Study Extension", In Latin-american Center for Informatics Studies Electronic Journal, 16(1), pp. 12-12, 2013.
- [5] R. Charette, "Why software fails", In IEEE Spectrum, 42(9), pp. 42-49, 2005.
- [6] E. Luna, J. Panach, J. Grigera, G. Rossi, O. Pastor, "Incorporating usability requirements in a test/model-driven web engineering approach", In Journal of Web Engineering, 9(2), pp. 132-156, 2010.
- [7] L. Rivero, T. Conte, "Improving Usability Inspection Technologies for Web Mockups through Empirical Studies", Proc. 25th SEKE, pp. 172-177, 2013.
- [8] L. Rivero, M. Kalinowski, T. Conte, "Practical Findings from Applying Innovative Design Usability Evaluation Technologies for Mockups of Web Applications", Proc. 47th HICSS, pp. 3054-3063, 2014.
- [9] F. Shull, J. Carver, G. Travassos, "An empirical methodology for introducing software processes", In ACM SIGSOFT Software Engineering Notes, 26(5), pp. 288-296, 2001.
- [10] A. Fernandez, E. Insfran, S. Abrahão, "Usability evaluation methods for the web: A systematic mapping study", In Information and Software Technology, 53(8), pp. 789-817, 2011.
- [11] L. Paganelli, F. Paterno, "Automatic reconstruction of the underlying interaction design of Web applications". Proc. 14th SEKE, pp. 439-445, 2002.
- [12] M. Allen, L. Currie, S. Patel, J. Cimino, "Heuristic evaluation of paper-based Web pages: A simplified inspection usability methodology", In Journal of Biomedical Informatics, 39(4), pp. 412-423, 2006.
- [13] F. Molina, A. Toval, "Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems", In Advances in Engineering Software, 40(12), pp. 1306-1317, 2009.
- [14] F. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology", MIS quarterly, pp. 319-340, 1989.
- [15] O. Laitenberger, H. Dreyer, "Evaluating the usefulness and the ease of use of a web-based inspection data collection tool", Proc. 5th International Software Metrics Symposium, pp. 122-132, 1998.
- [16] J. Nielsen, "Finding usability problems through heuristic evaluation", Proc. CHI'92, pp. 373-380, 1992.
- [17] M. Babar, D. Winkler, S. Biffl, "Evaluating the usefulness and ease of use of a groupware tool for the software architecture evaluation process", Proc. 1st ESEM, pp. 430-439, 2007.

AMBIT: Semantic Engine Foundations for Knowledge Management in Context-dependent Applications

Riccardo Martoglia

FIM Department, University of Modena and Reggio Emilia, I-41125 Modena, Italy,

E-mail: riccardo.martoglia@unimore.it

Abstract – Context-aware application and services proposing potentially useful information to users are more and more widespread; however, their actual usefulness is often limited by the “syntactical” notion of context they adopt. The recently started AMBIT project aims to provide a general software architecture for developing semantic-based context-aware tools in a number of vertical case study applications. In this paper, we focus on the knowledge management foundations we are laying for the Semantic Engine of the AMBIT architecture. The proposed semantic analysis and similarity techniques: (a) exploit the textual information deeply characterizing both users and the information to be retrieved; (b) overcome the limits of syntactic methods by leveraging on the strengths of both classic information retrieval and knowledge-based analysis and classification, ultimately proposing information relevant to the user interests. The experimental evaluation of a preliminary implementation in an actual “cultural territorial enhancement” scenario already shows promising results.

Keywords – context-aware applications; information retrieval; text analysis; semantic knowledge and similarity.

1. Introduction

Nowadays, we are constantly supported by ICT systems and applications that exploit ubiquitous services supporting different kinds of human activities. However, the availability of a large number of services can turn out to be confusing rather than useful, since the users are often overwhelmed by the large number of “proposals” which they are generally not able to consider thoroughly to find what they really need. To overcome this problem, many researchers have proposed to develop new applications with (or to incorporate in existing applications) context-awareness capabilities ([2, 4]). A context-aware application is one that “knows” the context in which the client is operating and possibly also the profile/characteristics of the user who is enjoying the corresponding service(s). Clearly, such knowledge must be gathered (often under real-time constraints), stored in well-organized fast-access data and information systems, and effectively exploited with the goal of delivering “personalized”

high-quality *context-dependent* services. This is far from simple; the main limitations of existing efforts lie in the limited notion of context they adopt, and especially in the almost complete absence of any attempt to model the semantics of the context.

This is the challenging scenario of the recently started AMBIT (Algorithms and Models for Building context-dependent Information delivery Tools) project¹ [5] a regional project co-funded by Fondazione Cassa di Risparmio di Modena and managed by the Softech-ICT research center. The main goal of the project is to study and implement a prototype software architecture for the development of *context-dependent applications and systems*, i.e., tools that provide users with services that are fully customized according to the context in which they operate. Preliminary steps will be the study of models, algorithms and data structures for the representation and manipulation of contexts. AMBIT will study and implement a very broad idea of context, including (among others) the modeling of the external environment, the users’ profile and the history of the actions performed by them.

The AMBIT software platform will eventually provide an API that can be personalized for the development of a number of vertical context-dependent applications. Several case studies have been identified by the project industrial partners; one of the main application scenarios, and the one which will be the reference in this paper, is the “cultural territorial enhancement” one: through both on-demand and proactive services, users of specific applications (including mobile ones) are empowered with precious “suggestions” pointing to the information (e.g. territorial activities, typical products descriptions, tourism information, etc.) which is the most relevant with respect to their profile and needs. A very simple example could be the notification of an event which is geographically close to the location of the user (say, a country fair with local farm exhibitors), which falls under the interests associated with his/her profile (e.g., gardening enthusiast). Another example could be the monitoring of tourists interests (e.g. through a dedicated mobile app where tourists could browse information on Emilia-Romagna typical products) and, based on their favorite browsed pages and/or on explicit queries asking for specific information/topics, the retrieval of the pages that best capture their interest.

(DOI reference number: 10.18293/SEKE2015-27)

¹<http://www.agentgroup.unimore.it/ambit/>

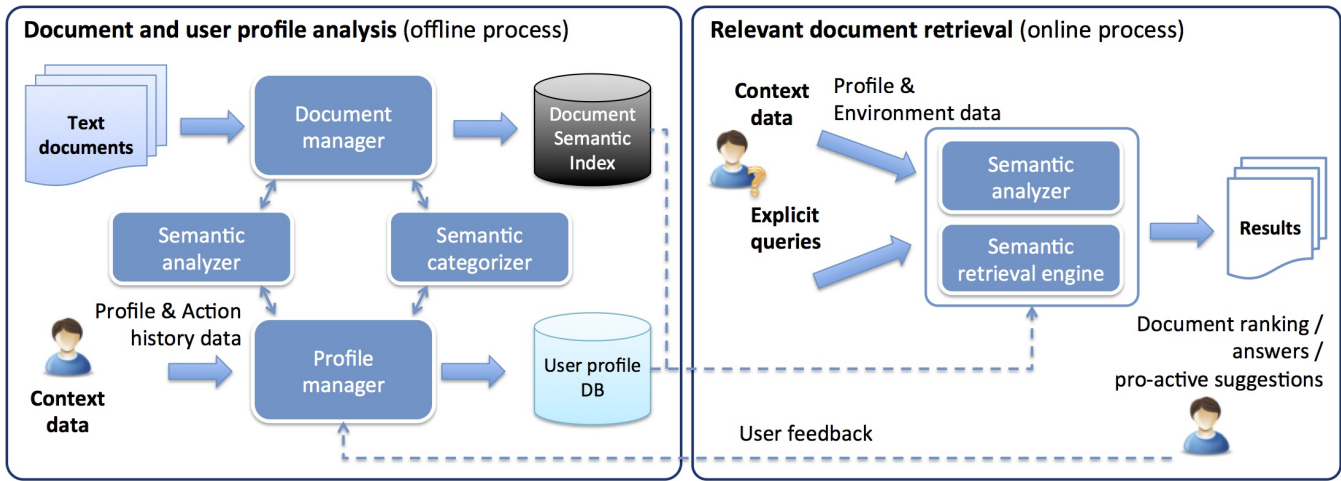


Figure 1. An overview of the AMBIT-powered Semantic Engine for information management

In order to achieve the AMBIT goals, several studies on complementary techniques and research fields will have to be performed. One of the most crucial among them, the key to provide “intelligent” suggestions and answers to users, is certainly to have powerful ways of managing available information and knowledge. In this paper, we focus on the foundations we are laying for the **Semantic Engine** of the AMBIT architecture, and, in particular, for its *knowledge management techniques* that are indeed one of the most challenging aspects of the AMBIT project and should be ultimately able, together with other AMBIT results, to deliver high-quality context-dependent information. The techniques we propose:

- take advantage of textual information, certainly the primary component of the documents that should be presented / suggested to users, and also one of the major information characterizing user profiles (think, for instance, to the contents of their browsing history, to the description of their interests, and so on);
- are completely flexible and designed to be easily applicable to the territorial enhancement scenario considered in this paper, but also to all the application scenarios involved in AMBIT (which also include, among others, context-aware advertising, smart help-desk problem solving, etc.).

More specifically, Figure 1 shows an overview of the main processes (and the related modules) of the semantic engine which will allow AMBIT-powered systems and applications to:

1. manage document and profile information (**Document and user profile analysis**, left part of Figure). This is done by extracting and indexing the associated semantics by means of ad-hoc *semantic text processing* and text classification techniques (described in Section 2), also exploiting external knowledge sources;
2. provide useful answers/proactive suggestions to the user (**Relevant document retrieval**, right part of Figure), by

retrieving the most relevant documents w.r.t. the user profile and/or query. This is achieved thanks to novel and specifically devised *semantic similarity* techniques (detailed in Section 3).

Section 4 shows preliminary but already promising results and the good effectiveness of the proposed techniques, by means of an experimental evaluation done on a small-scale actual territorial enhancement scenario. Finally, Section 5 concludes the paper also by briefly analyzing related works.

2. Document and user profile analysis

Document analysis. In this offline process, which is propaedeutic to the online document retrieval process (Section 3), the available documents are processed and the information which will be required in the actual retrieval is extracted, stored and indexed in an ad-hoc *Document DB* by a *Document manager* module (see left part of Figure 1). The input information are the text documents relevant to the specific scenario instantiation, including available web pages, product and service descriptions, and so on. For instance, in our territorial enhancement use case, these include descriptions of fairs and events which have been or will be held in the area, descriptions of typical products, details on forthcoming initiatives and activities, information on touristic points of interest, etc.

Since existing packages do not allow sufficient configuration and extension options, we preferred to design a custom-made *Semantic analyzer* tailored to the AMBIT environment. The analyzer performs several steps which are needed in order to extract the contents (and meaning) of the processed information, including: *Tokenization*, the terms of the different sections are identified and punctuation is removed; *Stemming*, the tokens are “normalized” and “stemmed”, i.e., terms are reduced to their base form (managing plurals and inflections); *POS (Part of Speech) Tagging*, the tokens are “tagged” with Part of Speech tags (i.e., nouns, verbs, ...); *Composite term identification*, pos-

a)	TERM	SYNS	IS_A	DEFINITION	IDF	DOC_LIST
	Expo	Exhibition, Exposition	Collection, aggregation	A collection of things (goods or works of art etc.) for public display	2.455	['D07542-3', ...]
	Parmesan		Cheese	Hard dry sharp-flavored Italian cheese; often grated	7.457	['D03522-3', 'D08654-2', ...]
b)	CLASS	IS_A	DEFINITION	DOC_LIST		
	Automotive equipment	manufacturing& engineering	Companies that produce components for automobiles	['D04342', ...]		
	Viniculture	agriculture	Production of wines from the vines to the finished products	['D03265', ...]		
c)	DOC	TERM	TF	W		
	D00001-1	Ferrari	0.545	0.977		
	D00002-1	ModenaTour	0.210	1.131		
d)	DOC	CLASS	SCORE			
	D00001	Automotive equipment	0.645			
	D00002	Tour operator	0.410			

Figure 2. Sample portions of the extracted Document Semantic Index: global view for terms (a) and classes (b), per-doc view for terms (c) and classes (d).

sible composite terms (such as “production area” or “wine tasting”) are identified by means of a simple state machine and of POS tags information; *Filtering and enrichment*, terms are associated to additional information (such as definitions, synonyms, ...) extracted from the thesaurus (i.e., WordNet², [12]).

The extracted information will enable the retrieval of the most relevant information for the user in the online phase. For instance, by means of synonyms, documents about an “exhibition” will also be relevant to a query about an “expo”.

Moreover, a *Semantic categorizer* processes text in order to tag each document with appropriate subject classes; we adopt the text-centric Media Topic NewsCodes taxonomies and vocabularies provided by IPTC³, a well-known taxonomy offering a very good level of detail and coverage of the AMBIT topics. Each class tag has a score (the higher the score the more relevant the class is for the document). For instance, a document about the typical “Lambrusco” wine will presumably have “viniculture” among its highest scoring associated tags.

By applying batch document analysis to the document collection, the semantic index is automatically generated. Conceptually, it consists of a *global view* (all terms/classes together with their occurrences and additional extracted data, see Figures 2-a and 2-b for an excerpt) and a *per-document view* (terms/classes occurrences in each document with their statistics, Figures 2-c and 2-d). In particular, *DOC_LIST* is the list of the documents IDs in which each term/class occurs. Each occurrence is also associated to a weight reflecting its importance and meaningfulness in the text (*SCORE* for classes and *W* for terms, corresponding to the TF/IDF [14] model used in information retrieval). As we will see, this will allow the similarity functions of the Semantic Engine to draw useful knowledge from both the semantic and the classic text retrieval worlds.

User profile analysis. The *User profile DB* is populated and updated by the *Profile manager* each time a user connects. In particular, user context data may include *Profile data*, i.e. personal data, likings, preferences, etc. explicitly submitted by the user, *Environment data*, e.g. location data as extracted from

GPS sensors, time of day, ambient information such as lighting, noise level, etc, and *Action history data*, i.e. information about past user actions. This last kind of data is the one we mainly focus on in this paper and is particularly crucial for the semantic engine: it may include, for instance, past accessed documents (e.g. browsed from an ad-hoc AMBIT app), past actions performed on some partner’s website (e.g. about typical Modena products), and so on⁴. The intuition is that the documents from the user history can be analyzed in a similar way to the scenario documents, therefore exploiting all the power of the Semantic analyzer and categorizer in order to associate meaningful terms and classes to users (and thus enriching the user profile DB with information analogous to the one discussed for document analysis and shown in Figure 2). Due to their complexity, such analyses are performed offline and will be available for more accurately processing future requests from the same user.

3. Relevant document retrieval

This is the phase where users connect and receive the results which are relevant to their status or need (see right part of Figure 1). We encompass both the computation of proactive suggestions (based on context data) and the retrieval of “on-demand” personalized information to explicit user queries. For instance, a user could submit an explicit query about “Typical food stores” and the engine, also based on the user’s past actions (i.e. browsing specific food descriptions) and environment, will produce a list of nearby stores which (s)he could find interesting, possibly personalized on the basis of its preferences/browsing history. However, the retrieval process could also work without any explicit user input. As an example, based on the current Profile/Environment data and on the user profile DB, the platform could detect that a user interested in sport cars is traveling by train and is reaching a stop near an international car fair; therefore, a proactive suggestion pointing to the fair web page would be pushed to his mobile device.

⁴The information may be directly available from the websites’ logs or mobile application data or, where applicable, it may be indirectly derived by means of appropriate web tracking mechanisms ([1]).

²<http://wordnet.princeton.edu/>

³<http://www.iptc.org/site/Home/>

In all cases, the *Semantic retrieval engine* module has to analyze the profile, environment and/or the query, access both the Document DB and the User profile DB data and produce a ranking of the available documents. In case of an explicit query, the ranking is directly presented to the user; in case of a proactive scenario, the suggestion for the document(s) on top of the ranking is sent to the user. We also plan to manage *feedback* on the relevance/usefulness of the received results; the profile manager will update the user profile information with typical requests and result feedbacks in order to dynamically modify preferences and, thus, avoid unwanted suggestions.

The computation of the document ranking is based on ad-hoc similarity metrics:

- the similarity $TextSim$ between the main terms of the available documents (and their sections) and those associated with the user profile (e.g. past navigated documents), possibly including explicit query terms;
- the similarity $ClassSim$ between the document and user classes;
- additional similarities on other aspects coming from profile and/or environment data, such as explicit preferences or likings, current time and location, etc.

As anticipated in the past sections, the need of effectively and efficiently computing similarities between the terms/classes characterizing user profiles and documents is crucial in AMBIT. We will now deepen the discussion of these two similarities, which are in the focus of the paper and are certainly key to the semantic understanding and satisfaction of the user query.

Text similarity ranking. $TextSim(U, D)$ quantifies, given a user profile U and a document D , the similarity of the user profile w.r.t. the document on the basis of their associated terms $t^U \in U$ and $t^D \in D$: In particular, the computation of $TextSim$ between a given profile U and each possible D (i.e., each available document in the semantic index) involves the following steps:

1. considering each term in U and finding the most similar term or terms available in D by exploiting a *term similarity formula* $TSim$;
2. inducing a **ranking** of the available documents (on the basis of $TextSim$), thus predicting which documents are relevant and which are not w.r.t. U .

Equation (1) shows the text similarity formula: the similarity is given by the sum (defined in (2)) of all term similarities between each term in U and each term in D maximizing the term similarity with the term in D :

$$TextSim(U, D) = \sum_{t_i^U \in U} TSim(t_i^U, t_{\bar{j}(i)}^D) \cdot w_i^U \cdot w_{\bar{j}(i)}^D \quad (1)$$

$$t_{\bar{j}(i)}^D = \underset{t_j^D \in D}{\operatorname{argmax}} (TSim(t_i^U, t_j^D)) \quad (2)$$

where $w_i^U = tf_i^U \cdot idf_i$ and $w_{\bar{j}(i)}^D = tf_{\bar{j}(i)}^D \cdot idf_{\bar{j}(i)}$. In this way, each term contributes to the final similarity with a different weight. Moreover, the limitations of standard syntactical techniques are overcome by computing $TSim$ by means of

Equation (3), which considers **synonyms** (as extracted in the semantic index) and **semantically related terms**:

$$TSim(t_i, t_j) = \begin{cases} 1, & \text{if } t_i = t_j \text{ or } t_i \text{ SYN } t_j \\ r, & \text{if } t_i \text{ REL } t_j \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

Besides equal terms and synonyms, the formula provides a further case (*REL*) where the two terms are not equal or synonyms, nonetheless they are in some way related from a semantic point of view (i.e., broader/narrower terms etc.): such terms will contribute with a similarity of r , where $0 < r < 1$ is a user-defined fixed similarity value. *REL* is computed in real time by exploiting the relations between terms coming from the WordNet thesaurus and, more in detail, the method proposed in [8], a well established metric relying on the hypernym relations: for instance, “pasta” will result related with “dish” and “pizza”.

Class similarity ranking. In addition to document terms, the classes associated by the semantic classifier can also significantly help in order to retrieve useful documents. This is obviously true if both a user profile and a document are strongly characterized by a common IPTC class (e.g. “Motor car racing”); however, also documents about Ferrari cars and tagged with a similar class “Formula One” would be of interest. This is achieved through $ClassSim(U, D)$ which quantifies, given a user profile U and a document D and in the same philosophy as $TextSim(U, D)$, the similarity of the user profile w.r.t. the document, in this case on the basis of their associated IPTC classes $c^U \in U$ and $c^D \in D$:

$$ClassSim(U, D) = \sum_{c_i \in U} CSim(c_i, c_{\bar{j}(i)}^D) \cdot s(c_i) \cdot s(c_{\bar{j}(i)}^D) \quad (4)$$

$$c_{\bar{j}(i)}^D = \underset{c_j \in D}{\operatorname{argmax}} (CSim(c_i, c_j)) \quad (5)$$

$$CSim(c_i, c_j) = \begin{cases} -\ln \frac{\operatorname{len}(c_i, c_j)}{2 \cdot H}, & \text{if } \operatorname{len}(c_i, c_j) < Th \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

The similarity $CSim$ (eq. (6)) between two classes c_i and c_j is derived from the hypernym metrics exploited for term similarity [8, 9]: it is computed as a score which is inversely proportional to the length of the shortest path connecting the two classes in the IPTC hierarchy; in case the length exceeds a configurable threshold Th , the similarity is null. H is a constant representing the maximum depth of the hierarchy (5 for IPTC).

Ranking fusion. The rankings $\tau_{text}, \tau_{class}$ induced by Eqs. (1) and (4), respectively, on the documents D given a profile U , are eventually fused in a final fused ranking $\hat{\tau}$ through a linear combination method [13], exploiting both terms and classes contributions:

$$s^{\hat{\tau}}(D) = \sum_{\tau \in \{\tau_{text}, \tau_{class}\}} \alpha_{\tau} \left(1 - \frac{\tau(D) - 1}{|\tau|} \right) \quad (7)$$

where $|\tau|$ is the length of the ranking and $\alpha_{\tau} \geq 0$ is a preference weight (default is 1 for both rankings). Only documents which are part of both rankings will appear in the final

User req	Our Results			Typical retrieval baselines					
	Prec	Rec	F	Syntactic			Syntactic, no t.a.		
				Prec	Rec	F	Prec	Rec	F
U1	0.94	0.93	0.94	0.88	0.21	0.34	0.88	0.17	0.29
U2	0.90	0.93	0.92	0.90	0.87	0.88	0.24	0.34	0.28
U3	0.92	0.95	0.93	0.91	0.42	0.58	0.12	0.23	0.16
U4	0.94	0.89	0.91	0.78	0.31	0.45	0.67	0.23	0.34
U5	0.94	0.91	0.92	0.83	0.10	0.18	0.18	0.08	0.11
U6	0.92	0.93	0.92	0.82	0.24	0.37	0.12	0.13	0.13

Figure 3. Effectiveness analysis: precision, recall and F-measure (our results on the left, two baselines on the right).

ranking. Finally, note that, through α_{τ} , the two similarity formulas presented in this section can be combined in a completely flexible way, in order to make the Semantic Engine adaptable to the specific needs of different scenarios.

4. Experimental Evaluation

We will now present the preliminary results we obtained from an exploratory effectiveness evaluation we performed on a first prototype of the Semantic Engine, in the context of AMBIT. Together with the project partners, we considered a first simplified instantiation of the “cultural territorial enhancement” scenario; the document collection is composed of nearly 2000 documents about activities, tourism, food, exhibitions and fairs, manufactures, arts and events of the Modena and Emilia-Romagna area. Starting from this collection, which serves as a source of possible suggestions, we also considered different user profile requests (“users” in the following) simulating different kinds of interests/preferences. In this evaluation, being the effectiveness of the proposed analysis and similarity techniques the current focus, user information is strictly composed by an action history context (typically, 10-15 past navigated external documents witnessing their interests) and possibly by explicit query terms (2-8 queries). We will also assume a “stable” situation, where users and documents have been already automatically analyzed and their relevant terms and classes stored in the Semantic Index and User DB, respectively.

Among the considered users, we selected a set of 6 (U1-U6) as the most representative ones. For each one, we compared the output of the Semantic Engine with a “gold standard”, i.e. relevant answers manually selected from the collection by expert project partners, and assessed precision, recall, and F-measure (Figure 3, left part). The results are compared with two baselines simulating the results offered by typical retrieval engines: a syntactic retrieval method ignoring synonyms, related terms and class information, and a syntactic method also ignoring text analysis (composite terms identification, stemming, etc.).

As we can see, the precision and recall levels achieved by the semantic engine are generally very satisfying: all users widely benefit from the proposed semantic features. Let us now analyze the results in detail. Typical terms and classes associated to U1 involve generic “tourism information”: documents about

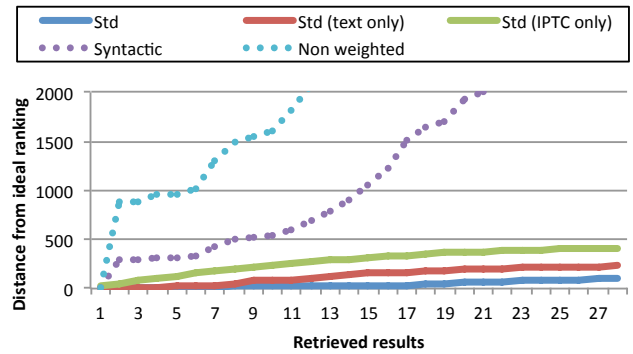


Figure 4. In-depth effectiveness analysis for U5: distance from optimal ranking

restaurants, hotels and tour operators are correctly retrieved by the engine due, for instance, to the similarity of the “tourism” “catering” and “accommodation” IPTC classes and of the contained terms (recall of 93%); on the other hand, they do not necessarily contain the same terms present in the user profile, thus syntactic techniques are not able to identify most of the relevant documents (recall of 21% or lower). Text analysis is also key to good results: for instance, the absence of stemming has a very negative impact on the second baseline recall for all users. Moreover, U2 and U3 are characterized by a high number of composite terms: for instance, U2 is mainly interested in “wine tasting”, while U3 in “picture card” expositions. These are two examples where ignoring composite term information (and including in the results irrelevant documents simply containing “tasting” or “picture”) can seriously affect precision (24% and 12% for second baseline, compared to 90% and 92% of the semantic engine).

The results, especially for U3-U6 which contain a larger number of terms, also benefit from synonyms, related terms and class management. For instance, U3 contains terms such as “exposition” which are correctly matched with several documents about “exhibitions” that, otherwise, would have gone unnoticed. User U4 is typically interested in “clothing” items and in motor car racing: documents about Ferrari Formula One “shirts” and “sweaters” are correctly retrieved mainly because those terms are found related to “clothing” in WordNet, and the documents themselves have also been associated to such IPTC classes as “motor car racing” (similarly to some browsed documents). Users U5 and U6 previously browsed mainly art and food documents, respectively; identifying the synonymy/relatedness of such terms as “church” and “cathedral”, “carving” and “sculpture” (U5) and “cheese” and “parmesan”, “food store” and “food shop” (U6) guarantees very good recall/precision (over 90%), differently from the baselines.

Finally, we deepened the effectiveness analysis by considering the actual rankings of the retrieved documents. Being the number of potential suggestions very high, it is essential to evaluate whether the best suggestions are returned in the top positions (i.e. the weighting scheme is effective), especially for proactive cases. Figure 4 shows the normalized Spearman

footrule distance [6] between the retrieved and the ideal ranking for U5. As we can see, the curves of the semantic engine (“Std” in figure) are the lowest ones, meaning the least distance to the optimal ranking, while syntactic and non-weighted baselines are not effective in providing the best suggestions first. In particular, the fused ranking takes the best from the class and text rankings, which together seem able to well capture the user interests. For instance, taking U5 as a representative example, documents classified as “monument and heritage site” (IPTC) are deemed of interest to the art-focused user; text ranking also significantly contributes by promoting documents describing specific artistic sites such as a “clock tower”, a term with an high weight in the user profile. Due to lack of space we do not show detailed analyses for the other users, however we found that the good performance of U5 is fully representative of the others.

5. Concluding remarks

Several works in the literature have highlighted the benefits of managing context information and/or proposed techniques and applications exploiting context-awareness capabilities ([2, 3, 4, 7]). In particular, a few works are directed towards context modeling, representation, and effective handling, aspects of particular interest to AMBIT. For instance, [3] proposes to design a context management system which is not application-dependent, [7] proposes an architectural framework for context data management, while [15] reports the result of a study on various context modeling and management approaches. However, most of the approaches in the literature primarily focus on specific aspects such as external user information, and/or do not consider the semantics of the context.

The techniques presented in this paper will serve as the foundations of the AMBIT Semantic Engine and are designed to be a first step in overcoming these limitations while being general enough to support different application scenarios. More specifically, they are focused on exploiting the textual information deeply characterizing both the documents to be retrieved and the user information, taking into account the users’ “history”, i.e. past navigated documents and requests. The proposed semantic similarity techniques leverage on the strengths of both classic information retrieval and of knowledge-based and classification techniques, adapted and extended from different contexts, from information disambiguation [9] to the querying of heterogeneous information in digital libraries and PDMSs [10] and assisted software engineering [11].

AMBIT has just started and the presented approach is one part of the final picture. Future work will include the design of the other components of the whole architecture, the possibility to “personalize” the retrieved information, for instance by highlighting the sections which should be the most interesting to the user, the study of additional aspects of a user profile and the extension of the semantic framework with additional relevant similarity techniques based on them. Finally, an actual real-world experimentation will be performed in the upcoming

project evaluation phase in a number of case studies proposed by the project industrial partners in their field of expertise. This will help us in obtaining useful suggestions about the quality of the proposed techniques and their improvement, while, at the local level of our territory, the newly created context-dependent services is expected to help to improve the offer and to increase the volume of business of the partners.

References

- [1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The Web never forgets: Persistent tracking mechanisms in the wild. In *21st ACM Conference on Computer and Communications Security*, Nov 2014.
- [2] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2:263–277, 2007.
- [3] C. Bolchini, G. Orsi, E. Quintarelli, F. A. Schreiber, and L. Tanca. Context modeling and context awareness: steps forward in the context-addict project. *Bulletin of the Technical Committee on Data Engineering*, 34:47–54, 2011.
- [4] G. Cabri, L. Leonardi, M. Mamei, and F. Zambonelli. Location-dependent Services for Mobile Users. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems And Humans*, 33(6):667–681, 11 2003.
- [5] G. Cabri, M. Leoncini, and R. Martoglia. AMBIT: Towards an Architecture for the Development of Context-dependent Applications and Systems. In *3rd ICCASA Conference*, 2014.
- [6] P. Diaconis and R. L. Graham. Spearman’s footrule as a measure of disarray. *Royal Statistical Society Series B*, 32(24):262–268, 1977.
- [7] P. Falcarin, M. Valla, J. Yu, C. A. Licciardi, C. Frà, and L. Lamorte. Context data management: An architectural framework for context-aware services. *Serv. Oriented Comput. Appl.*, 7(2):151–168, June 2013.
- [8] C. Leacock and M. Chodorow. *Combining Local Context and WordNet Similarity for Word Sense Identification*, chapter 11, pages 265–283. The MIT Press, May 1998.
- [9] F. Mandreoli and R. Martoglia. Knowledge-based sense disambiguation (almost) for all structures. *Information Systems (Information)*, 36(2):406–430, 2011.
- [10] F. Mandreoli, R. Martoglia, W. Penzo, and S. Sassatelli. Data-sharing p2p networks with semantic approximation capabilities. *IEEE Internet Computing (IEEE)*, 13(5):60–70, 2009.
- [11] R. Martoglia. Facilitate IT-Providing SMEs in Software Development: a Semantic Helper for Filtering and Searching Knowledge. In *SEKE*, pages 130–136, 2011.
- [12] G. A. Miller. WordNet: A Lexical Database for English. *Communication of the ACM*, 38(11):39–41, 1995.
- [13] M. E. Renda and U. Straccia. Web metasearch: Rank vs. score based rank aggregation methods. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 841–846, 2003.
- [14] G. Salton and C. Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.
- [15] N. M. Villegas and H. A. Mller. Managing dynamic context to optimize smart interactions and services. In M. Chignell, J. Cordy, J. Ng, and Y. Yesha, editors, *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 289–318. Springer Berlin Heidelberg, 2010.

Documenting Implementation Decisions with Code Annotations

Tom-Michael Hesse¹, Arthur Kuehlwein¹, Barbara Paech¹, Tobias Roehm² and Bernd Bruegge²

¹Heidelberg University, Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{hesse, kuehlwein, paech}@informatik.uni-heidelberg.de

²Technische Universität München, Boltzmannstr. 3, 85748 Garching b. München, Germany
{roehm, bruegge}@in.tum.de

Abstract

Software developers make various decisions when implementing software. For instance, they decide on how to implement an algorithm most efficiently or in which way to process user input. When code is revisited during maintenance, the underlying decisions need to be understood and possibly adjusted to the current situation. Common documentation approaches like JavaDoc neither cover knowledge related to decisions explicitly, nor are they integrated closely with knowledge management. In consequence, decision knowledge is rarely documented and therefore inaccessible, especially when developers have left the team. So, effective maintenance is hindered. We have developed an annotation model for decision knowledge and integrated it with the knowledge management tool UNICASE. The approach enables developers to document decisions within code without tool switches to lower their documentation effort. Afterwards, maintainers can exploit the embedded decision knowledge and follow links to external knowledge. This paper presents the approach and evaluation results of a first case study, which indicate its practicability.

1 Introduction

During the implementation of software, developers make many decisions, e.g. on how to implement an algorithm most efficiently or in which way to process user input. This means to solve a decision problem, which comprises a set of alternatives and criteria to compare them [16]. A comparison of alternatives, like using an external library instead of programming an algorithm, can be made by considering expert knowledge, personal experiences and the context of the decision. So, a complex and large amount of knowledge is required to understand a decision problem in retrospect. We will refer to this knowledge as *decision knowledge*.

Over time, decision knowledge can erode easily [12]. As a result, most information needs of developers towards im-

plementation decisions cannot be satisfied sufficiently. This is shown in a study of Ko et al. at Microsoft with 17 software developer teams [13]. For instance, the question “Why was the code implemented this way?” could not be answered in 44% of the cases. The major reasons are that decisions either are documented within unstructured inline comments, are not documented at all or have to be inferred from external documents without links to code [15]. These reasons imply three requirements our approach has to fulfill.

First, implementation decisions are difficult to understand in retrospect, when no documentation structures are defined and unstructured inline comments are used. So, defined structures for decision documentation are required. Even frameworks like JavaDoc only provide limited capabilities for documenting decisions, as they focus on describing what was implemented, but not on the underlying decisions. But a defined documentation template for decisions often requires more than the decision knowledge, which is currently present. In consequence, a *structured, but incremental capture* of decision knowledge is required (requirement R1). Second, implementation decisions may concern code parts of different granularity levels, such as the usage of a particular operation or the purpose of an entire class. If developers cannot document decisions directly within the code, they either do not document decisions at all or have to interrupt their current implementation task and change to some external documentation tool. Therefore, decision knowledge should be *embedded within the code* for different levels of code granularity (requirement R2). Third, when decision knowledge is not linked to external documents like requirements or design diagrams, such decision-related external knowledge cannot be exploited easily. This again can cause high efforts and thereby hinders the assessment of implementation decisions by developers during system maintenance. Therefore, decision knowledge should be *linked to related external knowledge* within code (requirement R3).

The contribution of this paper is an approach, which adheres to these requirements. First, we propose a documentation approach based on code annotations, which is inte-

grated with knowledge management. Therefore, we derive appropriate annotations for decision knowledge from an existing knowledge model for decisions, and implement these annotations in Eclipse. The set of annotations covers many elements of decision knowledge and can be used in an incremental way without a static template. Second, we integrate our approach with the model-based knowledge management tool UNICASE [3]. This allows for links between annotations and external knowledge within UNICASE. Overall, our approach supports decision documentation within code for developers and makes decision knowledge explicit and exploitable during maintenance.

The remainder of this paper is structured as follows. Section 2 introduces background information and discusses related work. Section 3 describes our approach with a running example. In Section 4, we present results for a first evaluation of our approach. We summarize our insights in Section 5 and describe directions for future work.

2 Background and Related Work

In this section, we define *decision knowledge* in detail and introduce the *knowledge management tool UNICASE*. Then, we give a brief overview of *existing annotation approaches* for decision knowledge in code.

Decision Knowledge As defined in Section 1, *Decision knowledge* addresses all information required to understand a given decision problem with its context and rationale justifying the decision. Decision problems comprise a set of alternatives, which are compared by different criteria [16]. In practice, these criteria and the resulting rationale for the decision depend on various context aspects. For instance, constraints brought up by former design decisions or assumptions on the environment of the system shape decisions. Moreover, rationale for decisions might be influenced by time pressure in the project or personal experiences of developers [19]. As we have described in [17], many models exist that cover parts of this knowledge for different activities, for example in requirements engineering or design. However, none of these models is addressing implementation decisions, they do not support an incremental documentation and have only limited support for pre-defined links.

Due to these shortcomings, we decided to use a flexible documentation model as presented in [11]. It consists of a set of different decision knowledge elements, which may be aggregated for a decision incrementally over time by different developers. The model is depicted in Figure 1. The basic element is the *Decision*, which contains all related decision knowledge elements for one decision as *DecisionComponents*. Amongst others, *DecisionComponents* can be refined to a decision problem description as an *Issue*, to context information like an *Assumption*, to a solution description

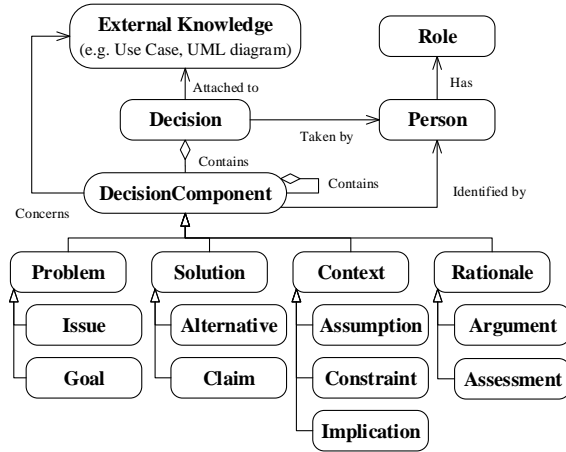


Figure 1. Documentation Model for Decisions

tion as an *Alternative*, or to a description of a rationale as an *Argument*. Decisions and their components can be linked to external knowledge like requirements specifications or design diagrams.

Knowledge Management Tool UNICASE Our annotation model is integrated with the model-based knowledge management tool UNICASE [3]. UNICASE is an Eclipse extension and provides an integrated model for system and project knowledge [5]. UNICASE offers a generic support for the collaborative editing of the underlying model elements based on the Eclipse Modeling Framework EMF [1] and the model versioning system EMFStore [2]. Moreover, it provides a variety of elements for documenting and structuring external knowledge, like use cases, UML diagrams or test protocols.

Existing Annotation Approaches In the last decades, only a few approaches were developed to document decision knowledge explicitly within code. Typically, they focus on rationale. As one of the first approaches, Lougher and Rodden introduced a system to annotate rationale in the source code of software using comments [15]. Then, a documentation is generated as a network of linked hypertext documents out of these comments. Whereas the approach claims to use a markup language for comments, no concrete proposals for well-defined structures for such a language are made. Moreover, the system is not integrated with external knowledge sources. So, this approach does not satisfy R1 and R3. Canfora et al. propose the “Cooperative Maintenance Conceptual Model” (CM²) [7]. It also structures rationale knowledge as a network of linked comments for analysis, design and implementation. Also for this approach well-defined structures for comments are missing. Moreover, the links between analysis and design artifacts

with the comments are not specified in detail. In consequence, also this approach does not fulfill R1 and R3. Burge and Brown present the system “Software Engineering Using RATIONale” (SEURAT), which is an Eclipse extension based on an ontology for rationale knowledge [6]. It can highlight existing rationale in the code via Eclipse markers as well as infer unresolved issues of inconsistencies. But the documentation of newly acquired rationale is not possible within the code, as the ontology has to be extended externally. Also, links to external knowledge are limited, as the approach focuses on linking rationale and code files. So, R2 and R3 are not satisfied. Other approaches enable developers implicitly to exploit decision knowledge by creating traceability links, e.g. between requirements and code. For instance, the approach of Cleland-Huang et al. [8] traces architectural significant requirements to code. This enables developers to reflect that a part of code realizes the linked requirements. However, such traceability links typically do not support incremental modifications and are not embedded within the code, so they do not fulfill R1 and R2. In summary, to the best of our knowledge no current approach realizes all three introduced requirements for decision documentation of implementation decisions.

3 Annotation Model for Decision Knowledge

Based on the documentation model for decision knowledge, we derived one annotation for each decision knowledge element and integrated the annotations with UNICASE. This implementation of our approach is available via an Eclipse update site [4]. In the following sections, we introduce a running example and describe how our model realizes the three requirements presented in Section 1.

Running Example To explain our annotation model, we will employ the decision on implementing a wizard instead of a dialog as example. This is a typical decision point when programming plug-ins for Eclipse. On the one hand, a wizard provides multiple pages for a guided user interaction, but typically requires multiple user actions for stepping through the pages. Moreover, it often implies complex data handling, when input checks for each page are performed. On the other hand, a dialog only offers a single page, so that a step-wise user interaction is not possible directly. However, a dialog typically requires less user actions, because it just consists of one page. Depending on the actual decision context, either a wizard or a dialog are more appropriate. We will refer to this decision in the following sections and enrich it with further information to demonstrate our approach.

Annotation Structure All annotations are mapped to a corresponding decision knowledge element of the docu-

mentation model (cf. Figure 1). An annotation can be used to either create a new decision knowledge element or link to an existing one. Each annotation contains several internal attributes to deal with references to external knowledge and persistent storage. The textual content that is to be documented by the annotation can be typed directly after the annotation itself by the developer, as depicted in Figure 2 using our wizard example. We established two different kinds of annotations with different functional complexity: core annotations and augmented annotations.

Core annotations represent a decision knowledge element in the documentation model, for instance an issue as @Issue or an alternative as @Alternative. We created one core annotation for each knowledge element given by the documentation model. *Augmented annotations* represent one or more decision knowledge elements with pre-defined attributes or relations. This is a shortcut for developers in practice to create decision knowledge elements by patterns, so that the manual documentation effort can be reduced. For instance, @Contra can be used to create a new argument as a child of the nearest DecisionComponent and to link the argument to this component as an attacking argument. In our wizard example, the @Contra-annotation is used to argue against the “dialog”-alternative (cf. Figure 2).

This structure of annotations suits an incremental and flexible use. For instance, if a @Decision statement is already given, a developer can simply add another annotation like @Alternative to document a newly arisen alternative during re-engineering for this decision. So, developers can complete and update the given documentation in case this decision knowledge is incomplete or outdated. Also, they are not forced to document a pre-defined default set of annotations for a decision. Our approach is extensible, as developers can define their own customized augmented annotations. Overall, this enables a structured and incremental documentation of decisions and thus fulfills R1.

Annotation Embedding To implement our annotation model, we created a new annotation parser within Eclipse, which makes the annotations available for use. Decision annotations can be used in any inline code comment, including JavaDoc, to annotate class and method declarations as well as any code part within the method body. All annotations are written directly into the code file, so that their textual contents are not lost when the code is stored in a code versioning system. Whenever a developer types an annotation,

```
// @Decision Implement input UI using a wizard
// @Issue Complex user input
// @Alternative Use a dialog
// @Contra Need for step-wise user guidance
```

Figure 2. Examples of Decision Annotations

options for creating or linking related decision knowledge are displayed by hovering over the annotation. In addition, our implementation in Eclipse allows to directly create new knowledge elements as children of the nearest decision that is found in the code before, as depicted in Figure 3. Moreover, developers can annotate elements of one decision in different comments on different code parts, as long as no other decision is inserted in between. This addresses the problem of different code granularity levels and enables a documentation of implementation decisions directly within the code, so R2 is fulfilled.

Integration with Knowledge Management UNICASE allows for relating decision knowledge elements with annotations to ingrate them into knowledge management. This integration requires that for each annotation in code a corresponding decision knowledge element in the management tool is created or linked. In consequence, all decision knowledge elements were added to the UNICASE model. Then, any other UNICASE knowledge element can be linked with the decision knowledge element corresponding to the annotation. However, an explicit link is needed to relate code annotations and decision knowledge elements in UNICASE. In our model, this is done by the AnnotationLink knowledge element as the parent knowledge element for CoreAnnotation and AugmentedAnnotation, which all three were added to the UNICASE model. The AnnotationLink provides a relation to the decision knowledge element and uses the ID of the Eclipse marker for linking to an annotation. In addition, the current revision of the code file and the decision knowledge element is stored. These revisions are updated, whenever a change in code or knowledge management impacts an annotation. So, a collaborative, distributed usage of annotations is supported. This mapping is depicted in Figure 4.

In our approach, developers are enabled to create, modify or delete both annotations and knowledge elements as summarized in Table 1. For new annotations, developers decide to either create a related decision element or link the annotation to an existing one. Then, annotations can be related to any further UNICASE elements representing the related external knowledge. Modifying annotations requires the corresponding decision elements to be updated,

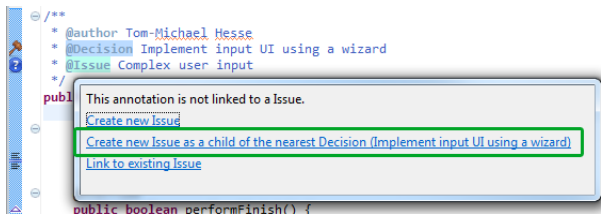


Figure 3. Create Elements using Annotations

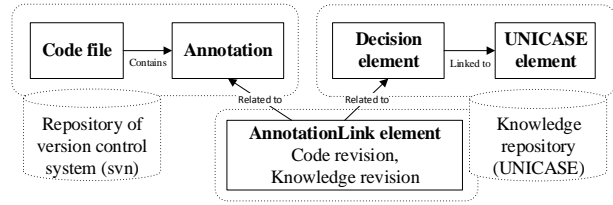


Figure 4. Relating Code Annotations and Decision Knowledge via AnnotationLink

whereas updates of decision elements in UNICASE may also require an annotation update. Considering our wizard example, a developer can document how the wizard class was embedded in the existing design. When annotations or their related decision elements are deleted, the related AnnotationLink is removed. If a deleted annotation was used to create a decision element, this corresponding element is also removed. If a decision knowledge element is updated or deleted, the related annotations also have to be updated or deleted within the code. However, this is currently not implemented due to restrictions and missing functionality in the employed Eclipse version 3.7. Through these actions, our annotation model enables developers to link any external knowledge consistently with annotations and thereby fulfills R3.

4 Evaluation

To investigate the practicability of our approach for other developers, we performed a first case study with students. We present its results in this section. However, this does not show the practicability of our approach in industry.

Context We performed a case study within a practical course for undergraduate students in computer science at Heidelberg University. Within the course, 7 participating students were grouped in two development teams in order to realize a software development project with identical project descriptions. Their task was to plan, implement and document an Eclipse plugin. We acted as the “customer”

Table 1. Impact of Developer Actions

Action	Performed on Annotations	Performed on UNICASE Elements
Create	Create new decision knowledge element or link existing one	No effect on annotation
Modify	Update decision knowledge element content, references, AnnotationLink	Update annotation, AnnotationLink
Delete	Delete decision knowledge element, AnnotationLink	Delete annotation, AnnotationLink

in both projects and provided an initial set of requirements as scenario descriptions, which were not changed during the project. Both projects were divided into three sprints lasted three weeks from mid February until the beginning of March 2015. For both teams, an initial tutorial for UNICASE and the code annotations was held to reduce the variability of competency concerning the annotations for the students. In addition, we provided textual explanations on how to use the code annotations to both teams. However, there was no mandatory rule for the students to use the annotations during implementation in order to get a realistic impression of the actual annotation usage. At the end of each sprints, the teams held a presentation to report on their current progress.

Research Questions and Method Our goal was to investigate the practicability of our approach referring to the research question: Is the annotation model and its implementation practicable to document implementation decisions? To evaluate this question, we build upon the Technology Acceptance Model (TAM) [9] to explore the actual use of our approach. TAM consists of three variables: *Ease of use* describes the degree to which a person expects the approach to be effortless, *usefulness* is defined as the subjective probability for a person to increase job or work performance and *intention to use* determines a persons' willingness to use the approach in the future. We assessed these variables with three anonymous questionnaires. Each questionnaire belonged to one sprint. They were answered by the students after each sprint presentation. We derived questions on using the annotations for each variable, as listed in Table 2. All questions were formulated as statements with defined answers in order to ensure the comparability of the students' responses. With statement #1 and #2, we distributed our investigation of ease of use on the creation and usage of annotations. Statement #3 and #4 address usefulness and intention of use for the entire approach. The answers represent a six point Likert scale [14], as this is an established approach in survey research. If the majority of subjects marks four or higher on the scale, we consider a statement to be accepted.

Table 2. Questionnaire Statements

No.	Statement	Variable
#1	It was easy to create decision elements with code annotations.	Ease of use
#2	It was easy to locate decision elements within the Eclipse Code Editor.	Ease of use
#3	Code annotations have been useful for the documentation of decisions.	Usefulness
#4	In the future I would use code annotations again to document decisions.	Intention of use

Table 3. Questionnaire Results

Sprint no.	Statement no.	Strongly disagree	Disagree	Rather disagree	Rather agree	Agree	Strongly agree	Not used, no answer	⊃ Disagree, Agree	Accepted
1	#1	0	0	0	0	1	2	4	0/3	yes
	#2	0	0	0	2	1	1	3	0/4	yes
	#3	0	1	0	3	1	1	1	1/5	yes
2	#1	0	0	0	1	2	1	3	0/4	yes
	#2	0	0	0	2	1	1	3	0/4	yes
	#3	0	1	0	0	1	3	2	1/4	yes
3	#1	0	0	1	0	3	2	1	1/5	yes
	#2	0	1	0	0	3	0	3	1/3	yes
	#3	0	0	1	0	2	2	2	1/4	yes
	#4	0	1	1	0	4	1	-	2/5	yes

Note, that only questionnaire 3 contained statement #4, as it addresses the overall experience with annotations during all sprints. For this statement, the “not used”-answer was not given. As the number of students does not permit to achieve statistical evidence, we also asked for rationale and comments in general and for each statement. This allowed us to collect as much individual feedback as possible.

Results The results from all questionnaires are presented in Table 3. Over time, more students used the code annotations, so that the sum of “Not used, no answer”-results slightly declines in sprint 3. Whereas the high number of “Not used, no answer”-answers especially in the first sprint provides only a limited support for the statements, no statement has to be rejected according to the number of rejecting answers. In consequence, this indicates that our approach is practicable for documenting implementation decisions with annotations. Multiple students pointed out in their feedback, that the approach was very useful to document decisions within the code in order to remember and reflect them. However, there was also a rejecting answer in the first two questionnaires concerning the usefulness of the annotations and several rejecting answers in the last questionnaire. This might be due to some errors in the integration of annotations and the code versioning system, which caused decisions to be represented at incorrect locations within the code. These errors partly are related to the employed Eclipse version 3.7 and were not entirely fixed during the course. Also, after trying our approach some students made proposals for functionality enhancements. For instance, they proposed to add keywords to annotations in order to create references to other decisions when typing the annotation.

Threats to Validity According to Runeson et al. [18], four different types of threats to validity have to be con-

sidered for our study. Concerning the *internal validity*, the students' knowledge was varying and they were not experienced in software engineering. To address this factor, we provided a tutorial for our approach and grouped the teams according to the students' subjective experience levels. However, missing experience could not be balanced completely. Concerning the *external validity*, the development projects had a rather small size regarding time, requirements, and team size. So, the evaluation of the usefulness of our approach might be affected. In addition, the results for the investigated student projects are incomparable to industry projects due to different project settings. However, Eclipse is a common tool in industry and also UNICASE has been used in an industry setting [10]. So, we believe that the usage through the students gives a first indication that our approach is useful in practice. Concerning *construct validity*, the questionnaires could have measured something different than TAM, as they were not evaluated prior to the study. However, we used typical questions for TAM. *Reliability validity* can be impacted by the fact, that the students knew we were investigating decision annotations. But this impact is unlikely to be high, as the investigators were not involved in the students' grading.

5 Conclusion and Future Work

This paper presented an approach to document implementation decisions using annotations in source code. The approach supports the structured and incremental capture of decisions within code without switching to a documentation tool. Moreover, external knowledge from knowledge management tools can be linked to annotated decisions. To the best of our knowledge no other approach addresses all of these requirements. The approach consists of an annotation model and is integrated with the knowledge management tool UNICASE. Evaluation results of a first case study were presented, which indicate the practicability of our approach.

In our future work, we will extend and improve our implementation of the annotation model. For instance, bugs with the integration of the code versioning system in the current implementation should be fixed and more code versioning systems (e.g., git) should be integrated. Moreover, we want to realize the functionality improvements acquired in the case study. Augmented annotations in our model could be extended, so that they can handle keywords as references on former or similar decisions. We also plan to execute further case studies in advanced practical courses and industry to overcome the shortcomings of the current study.

Acknowledgement This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future — Managed Software Evolution. We thank all students participating in our case study.

References

- [1] EMF. <http://eclipse.org/modeling/emf/> (05-2015).
- [2] EMFStore. <http://eclipse.org/emfstore/> (05-2015).
- [3] UNICASE. <http://unicase.org/> (05-2015).
- [4] Update Site for Decision Annotations. <http://svn.ifi.uni-heidelberg.de/unicase/0.5.2/ures/decdoc-features/> (05-2015).
- [5] B. Bruegge, O. Creighton, J. Helming, and M. Koegel. UnicaSe - An Ecosystem for Unified Software Engineering Research Tools. In *International Conference on Global Software Engineering*, pages 1–6. IEEE, 2008.
- [6] J. E. Burge and D. C. Brown. Software Engineering Using RATionale. *Journal of Systems and Software*, 81(3):395–413, 2008.
- [7] G. Canfora, G. Casazza, and A. De Lucia. A Design Rationale Based Environment for Cooperative Maintenance. *International Journal of Software Engineering and Knowledge Engineering*, 10(5):627–645, 2000.
- [8] J. Cleland-Huang, M. Mirakhorli, A. Czauderna, and M. Wieloch. Decision-Centric Traceability of Architectural Concerns. In *International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 5 – 11. IEEE, 2013.
- [9] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw. User Acceptance of Computer Technology: A Comparison of Two Theoretical Models. *Management Science*, 35(8):982 – 1002, 1989.
- [10] J. Helming, J. David, M. Koegel, and H. Naughton. Integrating System Modeling with Project Management - A Case Study. In *33rd Annual IEEE International Computer Software and Applications Conference*, pages 571–578. IEEE, 2009.
- [11] T.-M. Hesse and B. Paech. Supporting the Collaborative Development of Requirements and Architecture Documentation. In *3rd Int. Workshop on the Twin Peaks of Requirements and Architecture at RE2013*, pages 22 – 26. IEEE, 2013.
- [12] A. Jansen and J. Bosch. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109–120. IEEE, 2005.
- [13] A. J. Ko, R. DeLine, and G. Venolia. Information Needs in Collocated Software Development Teams. In *29th International Conference on Software Engineering (ICSE'07)*, pages 344–353. IEEE, 2007.
- [14] R. Likert. A Technique for the Measurement of Attitudes. *Archives of Psychology*, 22(140):1–55, 1932.
- [15] R. Lougher and T. Rodden. Supporting Long-term Collaboration in Software Maintenance. In *Conference on Organizational Computing Systems - COCS '93*, pages 228–238. ACM Press, 1993.
- [16] T. Ngo and G. Ruhe. Decision Support in Requirements Engineering. In *Engineering and Managing Software Requirements*, pages 267–286. Springer, 2005.
- [17] B. Paech, A. Delater, and T.-M. Hesse. Integrating Project and System Knowledge Management. In G. Ruhe and C. Wohlin, editors, *Software Project Management in a Changing World*, pages 161–198. Springer, 2014.
- [18] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering. Guidelines and Examples*. Wiley, 1st edition, 2012.
- [19] C. Zannier, M. Chiasson, and F. Maurer. A model of design decision making based on empirical results of interviews with software designers. *Information and Software Technology*, 49(6):637–653, 2007.

An Evaluation Study of Architectural Design Decision Paradigms in Global Software Development

Meiru Che, Dewayne E. Perry
Department of Electrical & Computer Engineering
The University of Texas at Austin, Austin, Texas, USA
meiruche@utexas.edu, perry@mail.utexas.edu

Abstract—Global software development (GSD) is considered as the coordinated activities of software development that are geographically and temporally distributed. The management of architectural knowledge, specifically, architectural design decisions (ADDs), becomes important in GSD due to the geographical, temporal, and cultural challenges in global environments. Based on our previous work on ADD management in localized software development (LSD), we present five ADD paradigms used for GSD projects with different organizational structures. We also investigate the benefits and the challenges of the ADD paradigms by conducting an evaluation of the paradigms using extensive archived semi-structured interview data from industrial GSD projects. We aim to provide a fundamental framework for managing ADD documentation and evolution in GSD, as well as offer useful insights into managing architectural knowledge in a global setting.

Keywords—architectural design decisions; global software development; documentation; evolution

I. INTRODUCTION

Global software development (GSD) is an increasing focus in the field of software engineering. It can be considered as the coordinated activities of software development that are not localized and centralized but geographically and temporally distributed [12]. Little attention has been paid to software architecting processes and software architectural knowledge management in the context of GSD. Similar to localized software projects, software architecting and architectural knowledge are important to support designing, developing, testing, and evolving software. We note, however, that in the global development of large complex systems, architecture plays an even more critical role in the structure of the project [11]. Therefore, managing and coordinating architectural knowledge such as architectural design decisions (ADDs) is a significant and also relatively new research problem in the context of GSD.

In our previous work on ADD management, we had an overall goal of providing a systematic approach that supports ADD documentation and evolution in a localized software development (LSD) context. Based on this, in this paper, we present and discuss five typical ADD management paradigms for global software projects, and we also conduct an evaluation on these paradigms using archived semi-structured interview data from industrial GSD projects to

investigate the benefits and the challenges of each paradigm in the GSD contexts. Since little work has been done on ADD documentation and evolution in GSD research and practice, we aim to provide a fundamental framework for managing ADD documentation and evolution in a global setting, and also provide better insights into architectural knowledge management for researchers and practitioners in GSD contexts in the field of software architecture.

To the best of our knowledge, our study is the first to provide ADD management paradigms in GSD projects and to support architectural knowledge management in global settings. Our study provides evidence that *managing ADDs in the GSD contexts reduces the complexity of coordination and integration among multiple distributed sites, decreases misunderstanding among different people, and also offers useful documentation for project planning and other management policies*. Our evaluation is also the first industrial investigation into the benefits and the challenges of global ADD management in practice.

II. BACKGROUND: ADD MANAGEMENT IN LSD

In order to capture the ADD set, we proposed the Triple View Model (TVM) to clarify the notion of ADDs and to cover key features in an architecting process [3]. The TVM is defined by three views: the element view, the constraint view, and the intent view. This is analogous to Perry/Wolf model's elements, form, and rationale but with expanded content and specific representations [18]. Each view in the TVM is a subset of ADDs, and the three views together constitute an entire ADD set.

Based on the TVM, we proposed the scenario-based ADD documentation and evolution method (SceMethod) [3], and we specified the element view, constraint view, and intent view through end-user scenarios, which are represented by message sequence charts (MSCs) [19]. By documenting all the possible ADDs and evolving these decisions with changing requirements, the SceMethod effectively helps us to make architectural knowledge explicit and to reduce architectural knowledge evaporation. Basically, we have four steps in the SceMethod to derive ADDs in a software project. For the sake of brevity, we will not discuss the detailed process of each step. We have the full illustration in [4].

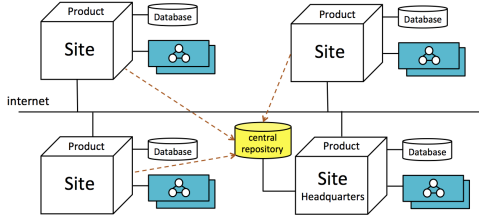


Figure 1. Product-based Paradigm in GSD (for network product)

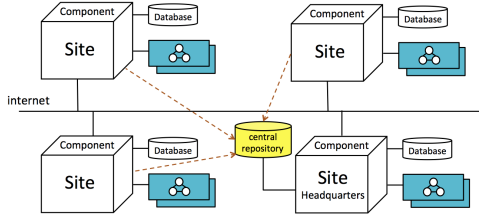


Figure 2. Product-based Paradigm in GSD (for single product)

III. ADD MANAGEMENT PARADIGMS IN GSD

In order to support ADD management in GSD projects, we proposed three strategies for managing ADDs in a distributed context, and discussed how distributed sites coordinate with each other to share and maintain consistent architectural knowledge. The three strategies for multi-site ADD management are federated strategy, client-server strategy, and incremental strategy respectively [5].

Given the foregoing discussion, we develop and discuss the following five paradigms for global software projects. Each paradigm adopts one strategy and is applied to one of the different organizational structures.

1) *Product-based Paradigm (Product-based Structure / Federated Strategy)*: We consider two cases for product-base paradigm, which are shown in Fig. 1 and Fig. 2.

In Fig. 1, the global organization works on a network product (such as the case in our evaluation in the next section), and each individual site is responsible for one individual/dependent product. In Fig. 2, the global organization works on a single product, then the product is decomposed into components and the different components are allocated to distributed sites.

We adopt the federated strategy to manage ADDs in the GSD projects with product-based structures. As shown in Fig. 1 and Fig. 2, each site manages ADD documentation and ADD evolution locally according to the TVM and the SceMethod. In addition, one of the global sites is selected as the headquarters and is to set up a central repository for recording and storing architectural decisions, which enables all the geographically distributed sites to share ADDs in the global context. These multiple sites have the access to the central repository, so that they can check in their local ADDs to the repository, read ADDs come from other sites, and even reuse ADDs from other sites when necessary.

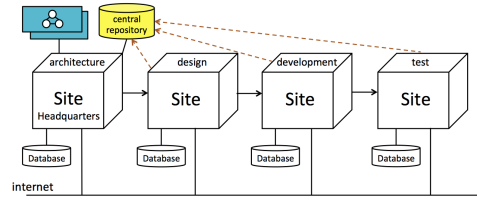


Figure 3. Process-based Paradigm in GSD

The headquarters with the central repository coordinates architectural knowledge in the repository and keep them consistent without conflicts. During the evolutionary process, the evolved ADDs from each site are also transferred to the central repository.

2) *Process-based Paradigm (Process-based Structure / Client-Server Strategy)*: For the process-based structures in GSD, the architecting process mainly occurs in the architecture phase, and all the other subsequent development phases are considered as the clients who access the ADDs derived in the architecture phase. Therefore, the client-server strategy provides us with suitable support for GSD projects with process-based structures.

In Fig. 3, we note that the architecting process is conducted in the site with architecture phase, relying on our TVM and SceMethod to derive the typical ADD set. Moreover, a repository is set up in the same site to manage architectural knowledge documentation and evolution. This repository is also regarded as a central repository among the global sites, and all the other sites have access to the repository for sharing and reusing ADDs in their specific development phases. In some cases, the subsequent development phases, such as design phase, may also come up with new architectural decisions as the process proceeds. However, we do not deal with this kind of exceptions for now, but only explore the general paradigms that are normally used in GSD.

3) *Release-based Paradigm (Release-based Structure / Incremental Strategy)*: The last two paradigms are both for GSD projects with release-based structures. We also discuss two formats in the release-based structures. One is for core-customized releases (which is the case in our evaluation in the next section), and the other is for incremental releases. Since different product releases are allocated to different sites, it is obvious that in the release-based paradigm each site derives its ADD set locally, and maintains ADD documentation and evolution in its local repository.

Figure 4 and Figure 5 show the ADD paradigms for the global projects with release-based structures. In Fig. 4, we can see that the ADDs from the core site are transferred to the customized sites. Besides combining the ADDs from the core site, each customized site has its local ADD management. Similarly, as illustrated in Fig. 5, each repository plays an important role in establishing a bridge to transfer architectural knowledge, which complies with

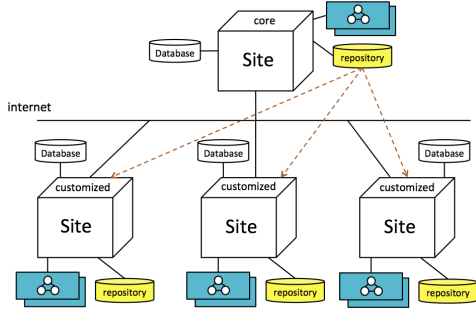


Figure 4. Release-based Paradigm in GSD (for core-customized releases)

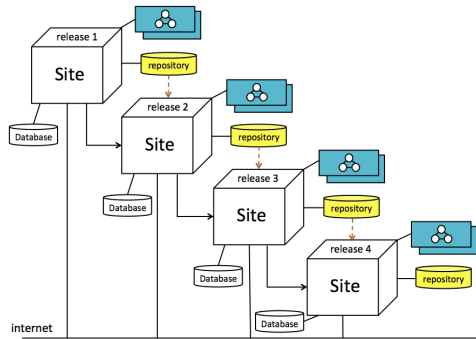


Figure 5. Release-based Paradigm in GSD (for incremental releases)

the mechanism in the incremental strategy. In the release-based structure, the core-customized releases or the multiple releases contain similar or even the same functionalities and product features, which implies that the ADDs derived from these different releases may have similarities as well. By adopting the incremental strategy in these two paradigms, each repository can serve as a reused ADD pool, and it is easy to combine, reuse, and modify ADDs.

IV. EVALUATION

In order to compare the ADD paradigms and evaluate whether they will bring benefits or introduce more challenges into global software projects, we investigate the aforementioned paradigms using extensive archived semi-structured interview data from Lucent Technologies [11], a telecommunications systems company with a number of geographically separated software projects. In our evaluation, we discuss the global projects in the following three aspects: degree of autonomy, resource requirement, and coordination complexity, which are the three factors largely influenced by the global settings.

A. Research Questions

We investigate the ADD paradigms by considering the following research questions:

RQ1: What are the benefits of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement,

and coordination complexity in different GSD organizational structures?

RQ2: What are the challenges of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement, and coordination complexity in different GSD organizational structures?

RQ3: How do the intent-related decisions and the evolutionary history of ADDs improve architectural knowledge management and project management in GSD projects?

B. Overview of the GSD projects

Twenty-seven interviews were conducted in six different organizations throughout Lucent Technologies. The interviews provided us with information about the project management and project evolution, as well as the distribution of work, and organizational and development situations.

We plan to look into the interview data from four organizations among the total ones. They respectively have different organizational structures. Each organization is briefly described here [11]:

Org_A produces a series of smaller products that are marketed together. Each product is developed by a single site, and all the different sites jointly provide a network product, including a manager component that ensures that all the others work together.

Org_B and *Org_C* build a very large telecommunications product together. They broke up their work into several process steps, and these steps are then used as handoffs among various locations.

Org_D has numerous sites. They produce software that is used for monitoring and managing networks. The basic product is built in USA, and the additional work for deliveries to particular customers is performed in Europe.

C. Analysis

To examine whether the ADD paradigms for GSD projects have visible benefits, or even bring in new challenges, we analyze the interview data from the four organizations above. We identify the characteristics of degree of autonomy, resource requirement, and coordination complexity for each organization, in order to obtain deep insights on the influence of ADD paradigms. We present our analysis in Table I.

As shown in Table I, we can see that in each organization, the interviews are conducted with several different roles in the software projects in order to provide GSD projects information from various points of view. Based on the organizational structures, we respectively adopt different ADD paradigms in each organization. Basically, Table I summarizes the organizations that we investigated from three aspects, which helps us understand how each ADD paradigm works in the corresponding organization.

Org_A has a high degree of autonomy, since each site works on an independent products/components. There are well defined interfaces and component functionalities

Table I
THE ANALYSIS AND COMPARISON OF THE GSD PROJECTS

	Org_A	Org_B, Org_C	Org_D
Role of Participants	Project Manager; Department Head; Software Developer; Tester; Software Architect	<i>In Org_B:</i> Senior Software Developer; Technical Manager; Quality Manager; Director <i>In Org_C:</i> Assistant Manager; Development Head; Software Developer	Technical Manager; Developer; Assistant Architect
Organizational Structure	Product-based Structure	Process-based Structure	Release-based Structure
Paradigm	Product-based Paradigm (for network product)	Process-based Paradigm	Release-based Paradigm (for incremental releases)
Strategy	Federated Strategy	Client-Server Strategy	Incremental Strategy
Degree of Autonomy	Needs network level testing; Has a coordinator for each release; Each site has a very close relation with the element manager; Needs lots of integration testing	The work in one site depends on that in another site; Needs to know the status of each site; Reports issues to other sites; Keeps consistent with requirements; A very clear agreement on handoff policy	A hybrid composition of component separation and process step; Core codes should be done before the customization; Needs to contact with customers; Needs to gather requirement for customization
Resource Requirement	Integration phase needs a lot of people; Every site needs experts; A defined process; Training & Tools	A well-defined software process; Defines the interface between sites; A documentation platform; Product architecture; Experts on each site; A stable plan on handoff policy and development process	Training; Documentation system; Expertise at custom site; Agreed plan for handoff
Coordination Complexity	Coordinates the combination; Defines interface across sites; Shares documents among sites; Phones & Emails & Meetings; Travelling	Phones & Emails & Meetings; Travelling; Different languages and time zones; Web is heavily used	Coordination with customers; Negotiations between customization people and core code people; Different languages, cultures, and time zones; Phones & Emails; Continuous Communication among different sites
Summary	<i>High</i> degree of autonomy; <i>Normal</i> resource requirement; <i>High</i> coordination complexity	<i>Low</i> degree of autonomy; <i>High</i> resource requirement; <i>Normal</i> coordination complexity	<i>Normal</i> degree of autonomy; <i>Normal</i> resource requirement; <i>High</i> coordination complexity

that contribute to the high autonomy. However, they do need to coordinate features across all the individual products/components and this need of being consistent on features does introduce a high degree of coordination complexity, for they need to coordinate when integrating the products/components to make sure everything works consistently. The following quotes show a few examples of the high autonomy and coordination in Org_A.

“We are doing the testing of the element manager in combination with all the different network elements.”

“Make sure the network management can manage the network elements and they also interwork.”

“What we will do is try to combine all those products together in a single network.”

As for Org_B and Org_C, they have multiple sites with different development phases of the project, and the work in one site depends on that in another site. Thus the main challenges for these two organizations are the high dependency between sites and the agreed handoff plan describing the points on what is to be handed off, and how and when to do so. In our investigation, we found that Org_B and Org_C have low degree of autonomy due to the dependency, as well as high resource requirement especially for a well-defined software process, the interfaces between sites, and a clear handoff plan. The following quotes describe the responses from different interviewees about their work.

“One feature was developed here in X and the other one in Y and I could say that we could not test our feature if we didn’t have their feature.”

“Well what we needed to know was if the planning that they had in X was just a little bit in front of our planning since that we didn’t have to lose time because we just had to wait for them to finish.”

“Here I have sort of a pretty well-defined software process.”

Org_D has the release-based structure, and it contains one site responsible for the core code, as well as all the other sites for the custom codes. Basically, the customization site obtains the core code from the core site, and customizes the code according to the requirements from local customers. From our investigation on the interview results, Org_D normally requires high coordination, since there are much negotiation between the customization site and core site, as well as coordinations with various customers. We can see more examples from the interview.

“once the allocation has been made of where different processes are going to be developed, there is a need for continuous communication coordination.”

“there is much negotiation between customization people and the core code people, but what most of the time happens then is that an expert from our site takes a look at it.”

D. Results

In this section, we look into what kinds of benefits and challenges will be brought in when adopting the ADD paradigms in these organizations with different structures, and answer the research questions.

RQ1: What are the benefits of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement,

and coordination complexity in different GSD organizational structures?

We adopt the product-based paradigm in Org_A. In this paradigm, each site manages its ADDs locally, and the multiple sites share their ADDs in a central repository. This enables us to keep architectural knowledge consistent among different sites, and decreases the resources which are used for product training and documentation. Most importantly, the explicit architectural knowledge provides us with more detailed and clearer architecture issues and specifications, which reduces the complexity of coordination and integration among different sites.

The main benefit provided by the process-based paradigm for Org_B and Org_C is that it is easier to establish a well-defined architecture and software process for the organization with process-based structure. Moreover, the ADDs capture key constraint decisions, which leads to a clear and consistent agreement on the handoff specifications. The recording and sharing of ADDs largely decreases the cost of resource requirement in the global organizations, as well as the issues in the dependency among sites.

Similarly, Org_D with release-based paradigm for the ADD management has high coordination complexity. However, the ADDs can be kept up-to-date and consistent with changing requirements from customers by using our TVM and SceMethod, which provides the core site and the customization sites with consistent project information, and decreases the negotiation between them. Moreover, ADDs offer a agreed plan for handoff policy used between the core site and the customization sites.

Overall, we find that ADD paradigms make ADDs explicit, and the pre-written architectural knowledge decreases misunderstanding in the global development context. In the meanwhile, the communication among different sites is in consistency from the beginning of the project, which reduces the degree of intensive coordination across the multiple sites.

RQ2: What are the challenges of each ADD paradigm regarding the aspects of degree of autonomy, resource requirement, and coordination complexity in different GSD organizational structures?

The main challenge when adopting the ADD paradigms in these global organizations is that more resources are required due to the ADD repositories. For Org_A with the product-based structure, the headquarters site has to set up a repository for storing ADDs, and this would increase the resource requirement but will not influence a lot. We have the similar problem in Org_B and Org_C with the process-based structure, i.e., one ADD repository needs to be set up at the headquarters site. However, for Org_D with the release-based paradigm, each local site needs an ADD repository, which takes up more requirements for hardware and software resources in the entire global project. The other challenge is that the access to ADD repositories among multiple sites also increases the coordination complexity in

the global organization.

RQ3: How do the intent-related decisions and the evolutionary history of ADDs improve architectural knowledge management and project management in GSD projects?

In our TVM and SceMethod, we can document the intent-related decisions and also update ADDs when software requirements change. This is consistent when we investigate the GSD project contexts. For the global settings, the ADD paradigms collect the intent from stakeholders located at different sites. During the process of integration and combination, which happens a lot in Org_A and Org_D, the stakeholders are more likely to have consistent architectural knowledge. Therefore, misunderstanding and negotiation among different sites and people are much decreased.

In addition, keeping the evolutionary history of ADDs in the global organizations helps the project maintain the documentation system in GSD settings and reduce the inconsistent issues. Specifically, in Org_D the evolutionary history of ADDs provides us with effective way to track the changes of the requirements, thus providing a better control on the customized requirements from customers, as well as the changing features under each release.

E. Threats to Validity

1) *Construct validity*: We select to evaluate the benefits and the challenges on introducing the ADD paradigms into the GSD projects. Specifically, we focus on how these ADD paradigms affect the autonomy, the resource, and the coordination of the GSD projects. We believe that what we investigate in our evaluation are commonly used and considered in the ADD management, and thus provide good insights into the research and the practice of architectural knowledge management.

2) *Internal validity*: The primary threat to the internal validity of our evaluation is overlooking relevant problems in the extensive interview data of the GSD projects. This could affect our analysis on ADD management in global contexts. We controlled for this threat by focusing carefully on the specific organizational structures, i.e., the product-based, process-based, and release-based structures, and narrowing the interview data down to no more than five interviewees.

3) *External validity*: In our evaluation, we use the archived interview data from software industry to investigate the ADD paradigms. As opposed to formal experiments that generally have an emphasis on controlling variables, our evaluation analyzes the data through observations in an open and unmoderated setting. Our evaluation on the archived data may not generalize to other global projects. We controlled for this threat by observing the three most typical aspects that influence GSD projects, i.e., degree of autonomy, resource requirement, and coordination complexity.

V. RELATED WORK

The key concepts of the traditional view on software architecture are components and connectors [18]. Currently,

software architecture is viewed as a set of ADDs [14], [22]. The architectural decisions in the software architecting process are increasingly focused on by researchers and practitioners [10], [16], and ADDs are also considered to be a part of architectural knowledge [17]. In [9], a systematic review for architectural knowledge is presented, and different definitions on architectural knowledge and how they are relevant to each other are discussed as well.

Guidelines for documenting software architecture has been provided in [6], [13], however, those documentation approaches do not explicitly capture ADDs in the architecting process. Recently, many models and tools have been proposed for capturing, managing, and sharing ADDs, most of which are discussed and used within a localized software development context [23], [15] and [21]. A detailed comparison of these existing models and tools has been done in [20]. However, the existing models are hard to support architecture evolution very well [2].

With the increasing attention paid to GSD, ADD management should be able to effectively applied in a GSD setting as well. However, little work has been done on ADD management in the GSD. A few of general architectural knowledge management practices for GSD have been proposed and evaluated in [7] and [8]. Furthermore, a literature review has been done [1] to explore architectural knowledge in a GSD context, and six architectural viewpoints are defined to model GSD systems in [24].

Notably, ADDs have not been widely discussed and supported in GSD, and the aforementioned approaches do not address in detail how to capture, share, and evolve ADDs in a global software project. Our current study in this paper is significantly different from these prior studies by focusing on the ADD management in the global practice and by providing the specific ADD paradigms that can be adopted in the global software industry.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we develop and discuss five typical ADD management paradigms that can be widely used in GSD, and provide a high-level methodology on how to manage the documentation and the evolution of ADDs in the GSD context. We also investigate the benefits and the challenges of the ADD paradigms by conducting an evaluation on the paradigms using extensive archived semi-structured interview data from industrial GSD projects.

Our study is the first to provide ADD management paradigms in GSD projects and to support architectural knowledge management in global settings. In our future work, we plan to implement the ADD paradigms by providing tool support, and apply them to more GSD projects to investigate their impact on GSD contexts and environment.

REFERENCES

[1] N. Ali, S. Beecham, and I. Mistrík. Architectural knowledge management in global software development: A review. In *ICGSE*, pages 347–352, 2010.

[2] R. Capilla, F. Nava, and A. Tang. Attributes for characterizing the evolution of architectural design decisions. *Software Evolvability, IEEE International Workshop on*, 0:15–22, 2007.

[3] M. Che and D. E. Perry. Scenario-based architectural design decisions documentation and evolution. In *ECBS*, pages 216–225, 2011.

[4] M. Che and D. E. Perry. Managing architectural design decisions documentation and evolution. *International Journal Of Computers*, 6:137–148, 2012.

[5] M. Che and D. E. Perry. Exploring architectural design decision management paradigms for global software development. In *SEKE*, pages 8–13, 2013.

[6] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.

[7] V. Clerc. Towards architectural knowledge management practices for global software development. In *SHARK*, pages 23–28, 2008.

[8] V. Clerc, P. Lago, and H. v. Vliet. The usefulness of architectural knowledge management practices in gsd. In *ICGSE*, pages 73–82, 2009.

[9] R. C. de Boer and R. Farenhorst. In search of ‘architectural knowledge’. In *SHARK*, pages 71–78, 2008.

[10] J. C. Dueñas and R. Capilla. The decision view of software architecture. In *European Workshop on Software Architecture*, pages 222–230, 2005.

[11] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: dealing with distance in r&d work. In *GROUP*, pages 306–315, 1999.

[12] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE*, pages 188–198, 2007.

[13] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[14] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *WICSA*, pages 109–120, 2005.

[15] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool support for architectural decisions. In *WICSA*, pages 4–, 2007.

[16] P. Kruchten, R. Capilla, and J. C. Dueñas. The decision view’s role in software architecture practice. *IEEE Softw.*, 26:36–42, March 2009.

[17] P. Kruchten, P. Lago, and H. V. Vliet. Building up and reasoning about architectural knowledge. In *QOSA*, pages 43–58, 2006.

[18] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17:40–52, October 1992.

[19] D. M. A. Reniers. Message sequence chart: Syntax and semantics. Technical report, Faculty of Mathematics and Computing, 1998.

[20] M. Shahin, P. Liang, and M.-R. Khayyambashi. Architectural design decision: Existing models and tools. In *WICSA/ECSA*, pages 293–296. IEEE, 2009.

[21] A. Tang, Y. Jin, and J. Han. A rationale-based architecture model for design traceability and reasoning. *J. Syst. Softw.*, 80:918–934, June 2007.

[22] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[23] J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Softw.*, 22:19–27, March 2005.

[24] B. M. Yildiz and B. Tekinerdogan. Architectural viewpoints for global software development. In *ICGSE*, pages 9–16, 2011.

An approach for classifying design artifacts

Sébastien Adam, Ghizlane El Boussaidi, Alain Abran

Department of Software and IT engineering
École de technologie supérieure
Montréal, Canada

Abstract—Software designers have to deal with a large number of distinct software design artifacts (SDAs), including requirements, patterns, and tactics. This paper proposes a technique that systematizes the classification of SDAs, and a classification scheme (CS) which organizes the SDAs into a matrix, in a manner derived from the Zachman Framework for enterprise architecture. An instantiation of this CS is a traceability matrix called a software-structure map (SSM) that records the SDAs and their relationships. The approach is illustrated through the analysis of the Template Method (TM) design pattern as an example of a SDA.

Keywords—software knowledge management, software artifacts, multi-dimension analysis, decision support systems

I. INTRODUCTION

During the development of a software system, the software designers deal with numerous software design artifacts (SDAs) such as goals, concerns, requirements, and design patterns. A SDA can be characterized using some related SDAs and issues that may threaten the success of a project. For instance, a design pattern [1] is a SDA that is characterized by a rationale, a solution, some consequences, and trade-offs. Somehow, the SDAs constitute the assets that embody decisions and trade-offs applied during the project. Several approaches propose a process or a technique aiming at managing the software artifacts (e.g., [3, 7, 8]). These approaches usually focus on a subset of the artifacts involved in the development process and on a specific development perspective. However, there is a lack of works that support a methodical management of the SDAs and their relationships.

The SAM (Software Architecture Mapping) framework [9] was proposed to manage the accumulated knowledge related to software design in an integrated and systematic manner. SAM enables to: 1) relate the SDAs to their factors of influence; 2) offer support to use the relevant SDAs and to appropriately solve their related issues; and 3) keep track of the adopted arguments and resolved issues. The SAM framework relies on a knowledge base that is populated by creating a set of matrices called *software structure maps* (SSMs). A SSM is a matrix that organizes software design artifacts and their relations. It is built using a classification scheme that is derived from the Zachman framework [8].

This paper presents the proposed classification technique that systematizes the creation of the SSMs (see Figure 1). The technique uses the classification scheme (CS) of the SAM framework for classifying the SDAs according to their descriptions in the literature – see Figure 2 [1, 2, 3, 10]. The technique is illustrated through the analysis of the Template Method (TM) design pattern as an example of a SDA.

The contributions of this paper are: 1) reusable specifications of the SDAs and their relationships based on a uniform SSM format; 2) a systematic technique for extracting and structuring the SDAs using the SSMs; 3) a flexible technique to transform textual descriptions to networks of SDAs. This paper is organized as follows. Section II presents an overview of the proposed classification technique. Section III introduces a case study to illustrate the classification technique. Section IV presents the related works and section V presents conclusions and future works.

II. OVERVIEW OF THE CLASSIFICATION TECHNIQUE

Figure 1 presents the proposed classification technique which aims at creating a SSM by extracting the verbs and nouns for structuring the SDAs and relationships that constitute the description of a style, a design pattern, or a tactic. The resulting SSM is a matrix of traceability that records design knowledge (DK) about the problem and solution spaces of a software design. The SSMs should be managed as part of the DK. A SSM captures DK about direct or indirect relationships between SDAs; it supports analyzing as presented in [9] how the SDAs impact the capacity of the software design to satisfy targeted objectives.

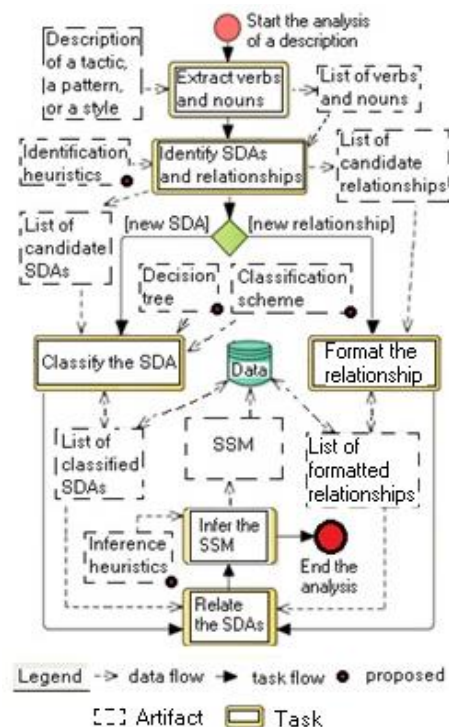


Figure 1. The classification technique of the SAM framework

	Rationale (Why)	Context (When)	Driver (What)	Structure (Which)	Behavior (How)	Allocation (Where)
Select objectives	needs, expectations, goals	organizational risks, politics, business model, situational factors	requirements, constraints, business rules	structures of domain objects	processes, activities, tasks, procedures	allocation of domain objects
Identify knowledge artifacts	architectural concerns	application domain standards, regulations, conventions	architectural properties	patterns and tactics	patterns of interactions	patterns of allocation
Define architectural artifacts	architectural design rationale	architectural risks, assumptions	scenarios	structural fragments	behavioral fragments	allocation fragments
Specify system artifacts	detailed design rationale	system's risks, assumptions	operation contracts	structures of modules	behaviors of components and connectors	allocations of elements
Describe architectural views	views descriptions	external entities, scopes, vocabularies, symbols	viewpoints	structural views	behavioral views	allocation views
Evaluate software structures	acceptance / assurance criteria	external and in-use metrics	internal metrics	structural evaluation records	behavioral evaluation records	physical evaluation records

Figure 2. The classification scheme of the SAM framework

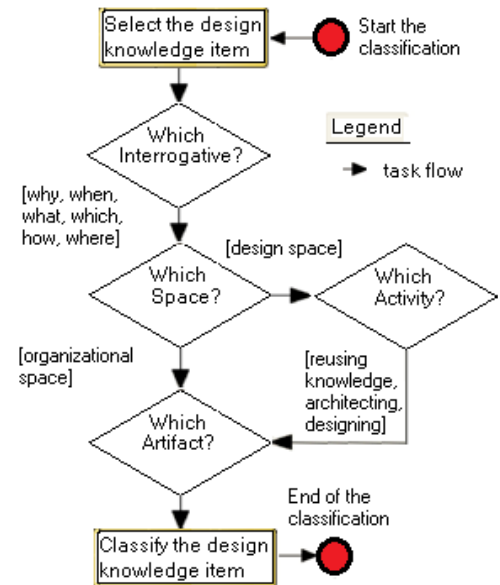


Figure 3. The decision tree of the SAM framework

A. The tasks of the classification technique

Six tasks constitute the proposed classification technique: extract verbs and nouns, identify SDAs and relationships, classify the SDA, normalize the relationship, relate the SDAs, and infer the SSM. The first and second tasks aim at identifying the candidate SDAs and relationships from the analysis of the description of a style, a design pattern, or a tactic using the identification heuristics. Then, the third and fourth tasks aim at classifying the SDAs using the decision tree and the classification scheme, and formatting the relationships using a list of formatted relationships. The fifth and sixth tasks aim at relating the SDAs and inferring the SSM by using the relationships and inference heuristics.

B. The proposed identification heuristics

For guiding the identification of the SDAs, we use in the SAM framework a set of identification heuristics. We consider that a SDA is: 1) less specific than implementation artifacts, i.e. implementation may be selected, within a particular technological context, to accomplish the intent of a SDA; 2) more enduring than implementation artifacts, i.e. a SDA should be described in a way that allows multiple implementations; 3) typically discovered or abstracted from practice and should have some correspondence with best practices such as styles, design patterns, and tactics; 4) coherent with more general or specific artifacts; 5) precise enough to be capable of analysis; and 6) related to one or more SDAs.

C. The proposed classification scheme

Figure 2 presents the proposed classification scheme (CS) of the SAM framework. The CS organizes the SDAs extracted from our analysis of the descriptions of styles, design patterns, and tactics. The CS captures the SDAs about the design problem and solution spaces, and about explicit or implicit relationships between the SDAs. The CS captures only the SDAs that influence the life cycle of a system.

The CS organizes the SDAs into a matrix that is based on the Zachman Framework for enterprise architecture [8]. The matrix classifies the SDAs according to their descriptions and relationships, as described in [1, 2, 3]. More specifically, the CS uses a matrix where the rows represent the activities of the software design process and the columns, the interrogatives (why, when, what, which, how, and where). The outcomes of the following activities occupy the row labels: select the objectives, identify the knowledge that has been successful in achieving similar objectives, and define, specify, describe, and evaluate the software architecture. The problem space is split into the interrogatives why, when, and what. The rationale (WHY issues) provides reasoning about the problem. The context (WHEN issues) describes the environment and hypotheses that influence the solution space. The drivers (WHAT issues) define the problem. The solution space is split into the interrogatives: which, how, and where. The domain objects and architectural elements have roles (WHICH issues) in realizing the solution. The execution of their behaviors (HOW issues) at the assigned locations (WHERE issues) shall satisfy the objectives for which a SSM is done. The SDAs in the top row of Figure 2 define the problems and solutions from an organizational perspective. The ones in the five lower rows do the same from a design perspective. Each lower-row contains artifacts for refining the interrogatives of the row that is above it, from the general objectives to the specific system artifacts.

D. The proposed decision tree

We propose to use the decision tree in Figure 3 for classifying the SDAs, and the following questions for supporting the classification task. The questions begin with the prefix “Does the SDA describes?”. Each question relates to one of the four main questions presented in the decision tree: which interrogative, space, activity, and artifact best render the meaning of the SDA in the context of a SSM?

- Which interrogative?
 - why: "... a reasoning for the SSM?"
 - when: "... a contextual information for the SSM?"
 - what: "... a target for a solution?"
 - which: "... the element of a solution?"
 - how: "... the behavior of an element?"
 - where: "... the allocation of an element?"
- Which space?
 - organizational: "... the organizational space?"
 - design: "... the design space?"
- Which activity?
 - reusing knowledge: "... an information that is part of the design knowledge base?"
 - architecting software: "... an information about a design fragment?"
 - designing software: "... an information about a design structure?"
- Which artifact?
 - use the SDAs' descriptions

E. The proposed SDAs descriptions

For classifying an artifact, we propose to use the SDAs described in Tables I to III. We extracted the proposed SDAs' descriptions from our review of the literature. Because of the lack of space, we describe only some SDAs that relate to the top four rows of the classification scheme.

TABLE I. THE DESCRIPTIONS OF SOME SDAs RELATED TO THE WHY INTERROGATIVE

Why: These SDAs provide reasoning for the SSM
Architectural / Design concern: an area of interest specified with respect to a goal in terms relevant for architecting / designing
Architectural rationale: a statement of reasons for a design fragment (e.g., isolate each layer from changes in other layers)

TABLE II. THE DESCRIPTIONS OF SOME SDAs RELATED TO THE WHAT INTERROGATIVE

What: These SDAs provide the targets for the solution space
Architectural property: a condition about a property of the elements or relations of a design fragment (e.g., performance)
Scenario: a description of how a software product should respond to a stimulus

TABLE III. THE DESCRIPTIONS OF SOME SDAs RELATED TO THE WHICH INTERROGATIVE

Which: These SDAs provide the elements of the solution space
Design pattern: a description of how the elements of a design fragment relate to each other in order to address a design concern
Structural fragment: a set of elements and relationships of a design fragment (e.g., instantiation of the template method)

F. The proposed relationships description format

We identified some relationships between the SDAs from the literature [1, 2, 3, 4, 5, 6, 7, 10] – see Table IV. The SAM framework proposes to format each relationship using a unique identifier, a description of the relation, and the SDAs between which the relationship applies, as example:

Identifier	Description	SDA-to-SDA
Generalize	A SDA generalize another SDA	Structure-to-Structure

TABLE IV. THE FORMATTED RELATIONSHIPS OF THE SAM FRAMEWORK

Relationship	Description
Mandatory	A SDA requires the presence of another SDA
Optional	A SDA optionally implies another SDA
Constraint	A SDA constraints another SDA
Encapsulate	A SDA encapsulates another SDA
Generalize	A SDA generalizes another SDA
Specialize	A SDA specializes another SDA
Realize	A SDA realizes another SDA

G. The proposed SSM's inference heuristics

Due to the lack of space, Table V presents only some of the inference heuristics we propose for inferring a SSM using the classified SDAs and the normalized relationships. These inference heuristics aim at controlling the level of cohesion between the SDAs of a SSM. Only one SDA drives the cohesion of the SSM. All SDAs within a SSM shall be cohesive with the driver SDA.

TABLE V. THE INFERENCE HEURISTICS FOR THE SDAs RELATED TO THE WHY INTERROGATIVE

SDAs	Inference heuristics
Architectural concern, Design concern	- Part of the design knowledge base - Describe concerns for the SSM's design space - Influence all SDAs of a SSM's design space - Relate to a goal in the SSM
Architectural rationale	- Set rationale for elements of a design fragment - Relate to an architectural concern in the SSM
Design rationale	- Set rationale for elements of a structure - Relate to a design concern in the SSM

III. CASE STUDY - APPLYING THE CLASSIFICATION TECHNIQUE

This section presents an overview of the case study selected for applying the classification technique of the SAM framework. We analyzed the descriptions of multiple architectural tactics in [3], design patterns in [1], and architectural style in [2] for creating their SSMs using the proposed classification technique.

A. Mapping for the Template Method design pattern

Table VI presents the SSMs of the Template Method (TM) design pattern described in [1]. The TM design pattern is used for providing reusability and extensibility of algorithms in object-oriented software. It aims to implement the skeleton of an algorithm in a base class, and calls primitive methods that subclasses override to provide concrete behavior. The base class interface declares the algorithm as a template method, which calls abstract primitive methods that represent the algorithm's variation points. The subclasses implement the primitives to specialize the algorithm. As a result, the algorithm's structure is written only once and indirectly specialized in subclasses, which reduces duplication of code and enforces class interface stability. Also, the template method allows the addition of instrumentation in the base class, and lightens users' duty since it is no longer required to call a primitive.

TABLE VI. THE SSM OF THE TEMPLATE METHOD DESIGN PATTERN

SDA	Description
Concern	Avoid code duplication
Concern	Control subclasses extension
Concern	Localize changes
Concern	Prevention of ripple effect
Rationale	Fix the steps of the algorithm and ordering
Rationale	Let subclasses define the steps of the algorithm
Rationale	Maintain the algorithm's structure
Rationale	Limit extension points
Rationale	Provide default behavior
Rationale	Control access to the operations
Situational f.	Multiple kinds of primitive operations
Convention	Naming convention
Symbol	UML notation
Property	Object-oriented paradigm
Property	Object-oriented programming language
Property	Reusability
Property	Extensibility
Operational.	Define an abstract base class
Operational.	Define a template method
Operational.	Define a concrete child class
Operational.	Define hook operations
Viewpoint	Class diagram
Viewpoint	Sequence diagram
Pattern	Template Method
Tactic	Abstract Common Services
Fragment	Class library
Structure	Abstract class definition
Structure	Concrete class definition
Behavior	The TM controls the order of execution
Behavior	The hook operations do nothing by default

IV. RELATED WORKS

To take full advantage of the accumulated design knowledge, the designers need frameworks and tools not only to manage this knowledge but also to relate it to the decisions taken and artifacts produced during the design activity. However, most of models, methods, and tools provide limited views into this knowledge base [2, 5, 6, 7, 10]. Many approaches were proposed to support the design process [3, 5, 6, 7], but few approaches support the designers in managing and keeping track of the accumulated knowledge during the design process. One of the most used approaches is the Attribute-Driven Design method (ADD) [3]. The focus of ADD is the process of architecting systems in order to satisfy a set of quality attributes and to manage tradeoffs between these attributes (quality dimension). Our approach can be used to analyze and keep track of the artifacts and knowledge produced by the ADD.

Many architectural styles and patterns have been described and cataloged in the literature [1, 2, 3], but few approaches support the designers in extracting the design knowledge from textual descriptions. We believe that the proposed classification technique can be used to systematically analyze textual descriptions provided in the literature, organize the design artifacts, and to explicitly relate the artifacts used during the design process.

Finally, our work is closely related to Ovaska *et al.*'s work [5]. They proposed an approach to fully integrate quality requirements into the software design process. Their approach allows the architect to manage and track the quality attributes from the requirements specification to the architecture design. This approach focuses on finding styles and patterns using some quality attributes. While this is very useful, an architect still needs to keep track of the rationale, objectives and other constraints that led to choose these quality attributes. Our framework can be useful to manage these relationships into a SSM that relates in a finer-grained manner the artifacts of the problem space to the ones of the solution space, from the organizational goals to the specific system artifacts. We believe a SSM is a valuable artifact for providing an integrated view of the knowledge.

V. CONCLUSION

In this paper, we described a classification technique to populate a design knowledge base by extracting the software design artifacts and their relationships from the description of a style, a design pattern, or a tactic. We applied the classification technique for classifying the SDAs according to their descriptions and relationships, as described in the literature [1, 2, 3]. This work produced evidences that the multi-dimensional analysis approach introduced in [9] is a valuable step towards handling artifacts as an integrated set of factors of influence. The proposed classification technique can be customized to better support particular development process and systems' needs. In particular, the SAM framework may be adapted to sustain different CS. In the near future, we plan to propose a tool support and guidelines to support the process of creating a SSM, eliciting related arguments, and analyzing these arguments. The ultimate goal of this work is to build a reference model of SDAs and arguments linked formally and exploited by algorithms.

REFERENCE

- [1] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", A.-Wesley, (1995)
- [2] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., "Documenting Software Architectures – Views and Beyond", Addison Wesley, Boston (2003)
- [3] Bass, L., Clements, P., Kazman, R., "Software Architecture in Practice", Addison Wesley, Boston (2003)
- [4] Kim, S., Kim, D.K., Lu, L., Park, S., "Quality- driven Architecture Development Using Architectural Tactics", Journal of Systems and Software 82, pp. 1211-1231 (2009)
- [5] Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., Aho, P., "Knowledge Based Quality-driven Architecture Design and Evaluation", Journal of Info. and Soft. Tech. 52, 577-601 (2010)
- [6] Shahin, M., Liang, P., Khayyambashi, M.R., "Architectural Design Decision: Existing Models and Tools", In: WICSA/ECSA 2009, IEEE, Cambridge, pp. 293-296 (2009)
- [7] Parizi, R.M., Ghani, A., "Architectural Knowledge Sharing (AKS) Approaches: a Survey Research", Journal of Theoretical and Applied Information Technology, 1224--1235 (2008)
- [8] The Zachman Framework, <http://zachman.com/about-the-zachman-framework> (2008)
- [9] Adam, S., El-Boussaidi, G., "A multi-dimensional approach for analyzing software artifacts", 25th SEKE, June 27-29, Boston (2013).
- [10] Standard, I.: ISO/IEC 42010 Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems. ISO/IEC 42010, (2011)

A Novel Hybrid Approach for Diarrhea Prediction

Yongming Wang

Department of Computer Science & Technology
East China Normal University
Shanghai, China
ymwang819@gmail.com

Junzhong Gu

Department of Computer Science & Technology
East China Normal University
Shanghai, China
jzgu@ica.stc.sh.cn

Abstract—Accurate and reliable forecasts of diarrhea incidences are necessary for the health authorities to ensure the appropriate action for the control of the outbreak. In this paper, a novel hybrid model known as EEMD-GRNN is proposed to forecast the diarrhea incidences. The proposed approach first uses Ensemble Empirical Mode Decomposition (EEMD), which can adaptively decompose the complicated raw time series data into a finite set of intrinsic mode functions (IMFs) and a residue, which have simpler frequency components and higher correlations. The IMF components and residue are then modeled and forecasted using GRNN and the final prediction result can be obtained by these prediction results using the principle of ensemble. The proposed hybrid method is examined by predicting the monthly diarrhea cases number of children and adult located in Shanghai of China. The experimental results indicate that the proposed EEMD-GRNN model provides more accurate forecasts compared to the other ARIMA, single GRNN models and hybrid model (EMD-GRNN). Overall, the proposed approach was effective in improving the prediction accuracy.

Keywords—Diarrhea prediction; Ensemble empirical mode decomposition; Generalized regression neural network; Hybrid approach

I. INTRODUCTION

An accurate and timely diarrhea prediction is crucial for predicting future health events or situations such as demands for health services and healthcare needs. As a kind of common and important infectious disease, diarrhea has a serious threat to human health and leads to one billion disease episodes and 1.8 million deaths each year (WHO, 2008). Hence, a robust prediction model for diarrhea facilitates preventive medicine and health care intervention strategies, by pre-informing health service providers to take appropriate mitigating actions to minimize risks and manage demand [1].

Over the past couple of decades, there have been wide attempts to capture the relationship between the available information using some straightforward linear regression assumptions, for example, the autoregressive integrated moving average (ARIMA). However, currently there is no evidence to support the assumption that the relationship between the past and future of diarrhea is a perfectly linear one. Many recent studies focus on the use of machine learning techniques, such as artificial neural networks (ANNs), to build a prediction model. Unlike traditional statistical models, ANNs are data-driven models. They do not require strong model assumptions and can map any nonlinear function without a

priori assumption about the properties of the data, even though the underlying relationships are unknown or hard to describe. Related works have shown that machine learning techniques outperform many traditional models.

In this paper we develop predictors using generalized regression neural networks (GRNNs) [2], a special type of neural networks. GRNN has only a single design parameter and is simple and fast in training. When using GRNN for diarrhea prediction, the observed original values of prediction variables are usually directly used for building prediction models. However, many factors underlie the diarrhea such as seasonal variations. Due to the complexity of the diarrhea incidence, it is difficult to capture its non-stationary property and accurately describe its moving tendency.

Empirical Mode Decomposition (EMD) [3] is a kind of adaptive signal decomposition technique using the Hilbert-Huang transform and can be applied with nonlinear and non-stationary time series. However, EMD suffers from an intrinsic drawback—the frequent appearance of mode mixing. Fortunately, there exists an improved method called Ensemble EMD (EEMD) which makes up for the deficiency of EMD. Different from other traditional decomposition methodologies such as wavelet decomposition, EEMD is an empirical, intuitive, direct and self-adaptive data processing method created especially for non-linear and non-stationary signal sequences. Therefore, the EEMD has been widely used in many fields [4-6]. However, existing literatures regarding diarrhea prediction have not adopted EEMD processes, and this study will be to fill this gap.

In this paper, we introduce EEMD and GRNN to predict the monthly number of diarrhea cases. A novel hybrid prediction algorithm called EEMD-GRNN is proposed. The proposed approach was compared with the EMD-GRNN, single GRNN approaches and traditional time series models, such as ARIMA, thus demonstrating that the proposed model is substantially featured with an excellent prediction capacity. Moreover, in order to evaluate the performance of the proposed approach, the real world diarrhea datasets are used as an illustrative example.

The rest of this paper is organized as follows. Section 2 reviews related methods used in this paper which are EEMD and GRNN. The proposed model is described in Section 3. Section 4 presents the experimental results and the

effectiveness of the proposed methodology is discussed. Finally, Section 5 concludes the paper.

II. METHODOLOGY

A. Empirical Mode Decomposition

The basic idea of EMD is to identify the intrinsic oscillatory modes and to decompose original time series data into a finite and small number of oscillatory modes based on the local characteristic time scale by itself [3]. The decomposition is based on the following assumptions [6]: (1) the signal has at least two extreme-one maximum and one minimum; (2) the characteristic time scale is defined by the time lapse between the extreme; and (3) if the data are totally devoid of extreme but contain only inflection points, then they can be differentiated one or more times to reveal the extreme. Final results can be obtained by integration of the components. With the assumptions of decomposition, an original data series $X(t)$ ($t=1; 2, \dots, T$) can be decomposed in terms of the following sifting procedure. The detailed process of the EMD algorithm is shown as follows [3, 14-16]:

Step 1: Identify local extreme in the data $\{x(t)\}$.

All the local maxima are connected by a cubic spline line $U(t)$, which forms the upper envelope of the data. Repeat the same procedure for the local minima to produce the lower envelope $L(t)$. Both envelopes will cover all the data between them. The mean of upper envelope and lower envelope $m_1(t)$ is given by:

$$m_1(t) = U(t) + L(t) / 2 \quad (1)$$

Subtracting the running mean $m_1(t)$ from the original time series $x(t)$, we get the first component $h_1(t)$:

$$h_1(t) = x(t) - m_1(t) \quad (2)$$

The resulting component $h_1(t)$ is an IMF if it is symmetric and have all maxima positive and all minima negative. An additional condition of intermittence can be imposed here to sift out wave forms with certain range of intermittence for physical consideration. If $h_1(t)$ is not an IMF, the sifting process has to be repeated as many times as it is required to reduce the extracted signal to an IMF. In the subsequent sifting process steps, $h_k(t)$ is treated as the data to repeat steps mentioned above:

$$h_{11}(t) = h_1(t) - m_{11}(t) \quad (3)$$

Again, if the function $h_{11}(t)$ does not yet satisfy criteria for IMF, the sifting process continues up to k times until some acceptable tolerance is reached:

$$h_{1k}(t) = h_{1(k-1)}(t) - m_{1k}(t) \quad (4)$$

Step 2: If the resulting time series is an IMF, it is designated as $c_1 = h_{1k}(t)$. The first IMF is then subtracted from the original data, and the difference r_1 given by:

$$r_1(t) = x(t) - c_1(t) \quad (5)$$

The residue $r_i(t)$ is taken as if it were the original data, and we apply to it again the sifting process of Step 1.

Following the above procedures, we continue the process to find more intrinsic modes c_i until the last one. The final residue will be a constant or a monotonic function which represents the general trend of the time series. Finally we obtain:

$$x(t) = \sum_{i=1}^n c_i(t) + r_n, \quad (6)$$

$$r_{n-1}(t) - c_1(t) = r_i(t)$$

Where r_n is a residue. Thus, residue $r_n(t)$ is the mean trend of $x(t)$. The IMFs= $\{c_1(t), c_2(t), \dots, c_n(t)\}$ include different frequency bands ranging from high to low. The frequency components contained in each frequency band are different and they change with the variation of time series $x(t)$, while $r_n(t)$ represents the central tendency of time series $x(t)$.

B. Ensemble Empirical Mode Decomposition

The EEMD [17] is the inheritor of the EMD. EEMD defines the true IMF components as the mean of the corresponding IMFs obtained via EMD over an ensemble of trials and generated by adding different realizations of white noise of finite variance to the original signal $x[n]$. The added white noise can help extract the true IMFs, and can offset them via ensemble averaging after serving their purpose [8]. Therefore, this can substantially reduce the chance of mode mixing and represent a significant improvement over the original EMD. The process of EEMD decomposition can be demonstrated by the following steps:

- 1) Add a white noise series to the original time series dataset;
- 2) Decomposition the data with added white noise into IFMs using the EMD procedure;
- 3) Repeat the step 1 and 2 iteratively, but use different white noise each time
- 4) Obtain the ensemble means of corresponding IMFs as the final results.

C. Generalized Regression Neural Network

The GRNN [7] is a kind of radial basis function networks which is based on a standard statistical technique called kernel regression. A typical GRNN is organized using four layers, namely the input layer, the pattern layer, the summation layer, and the output layer. The hidden layer has radial basis neurons, while neurons in the output layer have a linear transfer function. A typical architecture of the GRNN is presented in Fig. 1. Given a sufficient number of neurons, GRNN can approximate a continuous function to an arbitrary accuracy [8].

Given m input-output pairs $\{X, Y\} \in \mathfrak{R}^n \times \mathfrak{R}^1$ and as the training samples, assume the original design of GRNN, that is, the number of hidden neurons is equal to the number of training samples. For a desired estimate of system output vectors Y , under the input vectors X , is achieved by a regression calculation. The procedure of the GRNN model can be represented as:

$$Y(X) = E[Y/X] = \frac{\int_{-\infty}^{\infty} Yf(Y, X)dX}{\int_{-\infty}^{\infty} f(Y, X)dX} \quad (7)$$

Where X is a d -dimensional input vector, Y is the predicted value of the GRNN model, $E[Y/X]$ is the expected value of the output Y , given the input vector X , $f(Y, X)$ is the joint probability density function of X and Y . When probability density function adopt Gaussian function, the network output function of Y given the vector X :

$$Y(X) = \frac{\sum_{i=1}^n Y_i \exp(-D_i^2 / 2\sigma^2)}{\sum_{i=1}^n \exp(-D_i^2 / 2\sigma^2)} \quad (8)$$

Where D_i^2 is defined as $D_i^2 = (X - X_i)^T (X - X_i)$, σ denotes the smoothing parameter, X is the input variable of the network, X_i is a specific training vector of the neuron i in the pattern layer.

A good performance for GRNN method depends on smoothing factor σ , which is very important in using GRNN for prediction and determines the generalization capability of the GRNN. The smoothing factor is only free (adaptive) parameter, apart from the input and output layer, involved in the designing of the network.

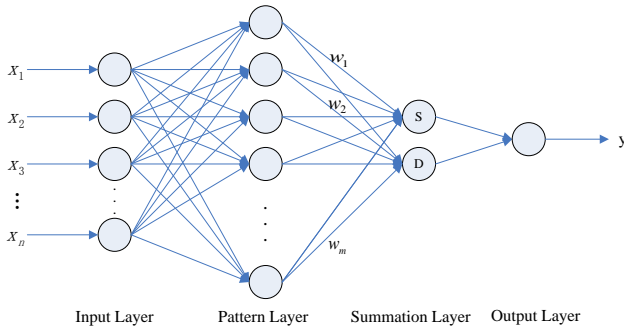


Fig. 1. Typical GRNN structure.

GRNN have several advantages [9], including: 1) it has one design parameter (smoothing factor); 2) it is easy to train since it is a one-pass algorithm; 3) it can accurately approximate functions from sparse and noisy data; 4) it can converge to the conditional mean surface by increasing the number of data samples; and 5) ability to model from a relatively small data set, and ability to handle outliers. It is these unique advantages that make us to choose GRNN as local models for each IFM.

From a time series prediction point of view, the purpose of GRNN is to define a function that produces outputs as close as possible to the actual values over the prediction horizon. Given a training set of T data points $\{(x_i, y_i) | x_i \in \mathfrak{R}^d, y_i \in \mathfrak{R}\}_{i=1}^T$, the GRNN try to construct a predictor function expressed by $y=f(x)$, where $f(\cdot): \mathfrak{R}^d \rightarrow \mathfrak{R}$ is the predictor function.

III. PROPOSED EEMD-GRNN APPROACH

Considering the aforementioned points in section II, in the current research the powerful combination of positive aspects

of EEMD and GRNN algorithm is presented to one-step-ahead diarrhea time series prediction problem. As shown in Fig. 2, the proposed EEMD-GRNN modeling framework is generally composed of the following three main steps:

Step 1: Decompose time series by EEMD

The original diarrhea time series are first decomposed into a finite and often a small number of intrinsic mode functions (IMFs) and a residue using EEMD technique (here the residual $r_{n+1}(t)$ also be considered as an IMF).

Step 2: Local-GRNN predictor construction

After the components (IMFs and a residue) are adaptively extracted via EEMD, each IFM component is modeled by an independent GRNN which are used to generate local predictor to forecast the component series respectively.

Step 3: Muti-local GRNN predictor ensemble

The forecasts of all IFM components are aggregated using another independent GRNN model, which model the relationship among the IMFs and the residue, to produce an ensemble forecasts for the original time series.

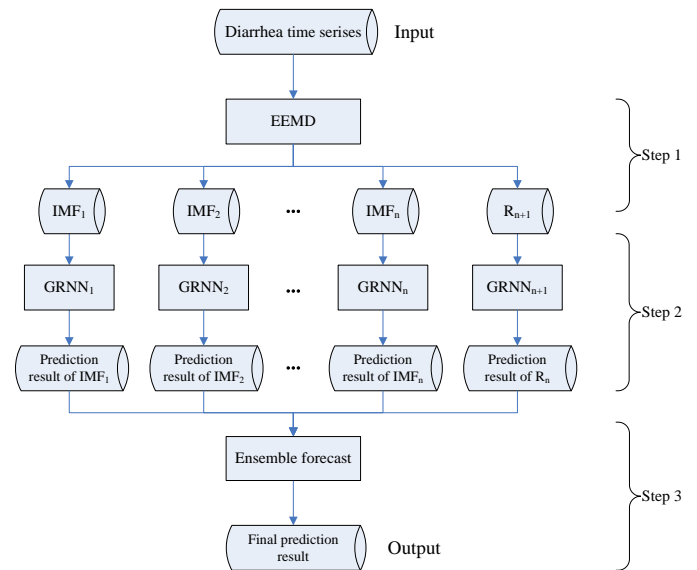


Fig. 2. The EEMD-GRNN modeling framework.

Several studies, for example [10, 11], have indicated that selecting model inputs is probably the most critical task for a time series prediction model, since it contains important information embedded in the data. The statistical approach to examine partial-auto-correlation function (PACF) of the time series was recognized as a good and parsimonious method in the determination of model inputs [12, 13]. So, in this study, the model inputs in the approach are mainly determined by the plot of PACF. After determining the relationship between input(s) and output(s), the input/output pairs can be constructed for each IFM component.

The performance of GRNN is mainly affected by the smoothing factor σ . There are no general rules for the choice of smoothing factor. In this study, the optimal smoothing factors for each local predictor are determined by the trial-and-error

method. Normalization required for neural network modeling in general is also included in our preprocessing. Thus, the inputs are normalized by the method of maximum and minimum normalization; after simulation, the corresponding estimate results are rescaled through the contrary process of the employed normalization method.

Through EEMD, different characteristics information of original time series can be displayed on different scales, and the proposed method can more fully capture the local fluctuations of raw data. Moreover, each IMF component has similar frequency characteristics, simple frequency components, and strong regularity, therefore allowing this model to reduce the complexity of local GRNN modeling and further improve GRNN prediction efficiency and accuracy.

IV. RESULTS AND DISCUSSION

In order to validate the effectiveness of the proposed EEMD-GRNN model, comprehensive experiments based on two real world diarrhea datasets were conducted. First, data description and performance criteria used in this study are presented and then the experimental results are reported. Finally, the result are compared and discussed.

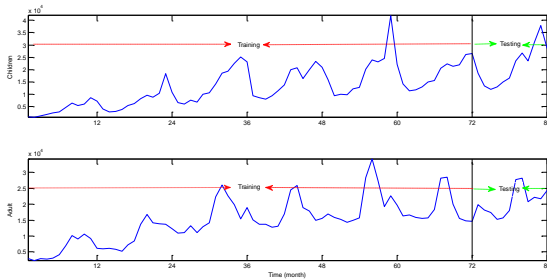


Fig. 3. Monthly diarrhea of children and adult from 2006(1)-2012(12).

A. Data Sets

In this study, the monthly diarrhea cases number data of children (0-15) and adult (>15) from 2006.01 to 2012.12 in Shanghai of China has been used. All data employed in this study are obtained from the Shanghai Municipal Center for Disease Control & Prevention. In particular, first, the data of the diarrhea of children are used to witness the whole process of the proposed method. In the same way, the corresponding prediction results of adult diarrhea are shown and further confirm the validity of the proposed method, accordingly.

There are in total 84 data points in each diarrhea dataset and the monthly series behavior is illustrated in Figure 3. The plot exhibits a permanent deterministic pattern of long-term upward trend with short-term fluctuations that are independent from one time period to the next. From Fig. 3, it can be seen that the series appear to be nonlinear, non-stationary in that the mean is increasing over time.

In order to testify the performance of the proposed prediction methods, the collected data is divided into two sets, training data and testing data. To achieve a more reliable and accurate result, a long period is served as the training period. Based on these considerations, the first 72 data points are used

as the training samples while the remaining 12 data points are used as the testing sample. The statistic characteristics of children and adult diarrhea in monthly time scale is tabulated in Table I.

To assess the forecast capacity of the EEMD-GRNN model, four indices for error forecast serve as the criteria to evaluate the prediction performance; they are mean absolute error (MAE), mean absolute percent error (MAPE), root mean square error (RMSE) and the coefficient of determination (R^2). MAE, MAPE and RMSE are measures of the deviation between actual and predicted values. The models with the smallest MAE, MAPE, RMSE and the largest R^2 are considered to be the best models.

TABLE I. STATISTICAL CHARACTERISTICS OF CHILDREN AND ADULT DIARRHEA FOR TRAINING AND TESTING DATA.

Indexes	Children		Adult	
	Training	Testing	Training	Testing
Max	41923.0000	37808.0000	34325.0000	28273.0000
Min	802.0000	11973.0000	2330.0000	15220.0000
Mean	13055.7500	21675.4167	14704.5556	20800.9167
SD	8069.2917	8214.7503	6892.3226	4296.5920

B. Prediction Results

According to the proposed hybrid EEMD-GRNN approach, in Stage 1, the original children diarrhea time series are decomposed into three independent IMFs (Illustrated in Fig. 4) and one residual employing EEMD technique, which exhibit a stable and regular variation. This means that the interruption and coupling between the different characteristics information embedded in the original data have been weakened to an extent. Thus, the local GRNN prediction model is easier to build.

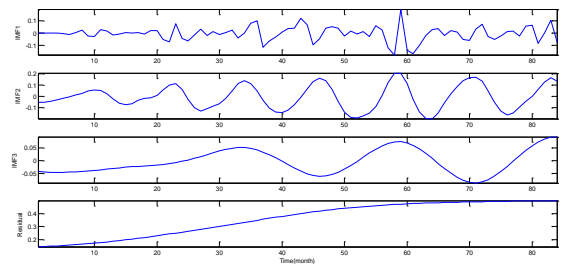


Fig. 4. EEMD decomposition result for children diarrhea.

After using EEMD to decompose the original children diarrhea data into three IMFs and a residue, these are then used to build the local GRNN prediction model for each IMF. In Stage 2, the relationship between the data of each IMF in the same frequency band should be identified prior to obtaining the prediction results of IMFs. The PACF is employed as a detector to determine the correlations between them. The lag orders of the autoregressive process of each IMF are two, four, five and five, respectively.

Based on the correlation between the data of each IMF, the input and the output pair vectors of the local GRNN model can be generated. Then the respective GRNN model is built and trained in terms of the input and the output vectors of the IMFs

and residue. For building the GRNN prediction model, the Matlab R2013a software package is adapted in this study and the optimal smoothing factor for each IFM is selected based on the corresponding minimum mean absolute error (MAE) on the out-of-sample testing samples. After that, the established GRNN model produces the one-step-ahead prediction results of each series of IMFs. In Stage 3, the final prediction results can be obtained using another independent GRNN model using the prediction results of each IFM as the input.

In order to reflect the model superiority, it is necessary to build other models to compare with the proposed model. Some other popular single prediction approaches recommended by recent works on time series prediction are selected as benchmarks. The benchmarks include time series techniques and artificial intelligence (AI) techniques. Amongst time series techniques, the autoregressive integrated moving average (ARIMA) models are adopted. For AI models, single GRNN is employed for the purpose. Furthermore, a hybrid learning approach with the EMD selected as decomposition method is also utilized. The simulation of this method is in general similar to the proposed model.

In the modeling of the single GRNN model like the EEMD-GRNN model, the input layer lags number has been determined using PACF and the smoothing factor are selected through the implementation of iterative optimization procedures.

In this study, the ARIMA model has three steps: model identification, parameter estimation, and diagnostic checking. The test time series data were processed by taking the first-order regular difference and the first seasonal difference to remove the growth trend and seasonality characteristics. We used the SPSS.19 statistical software to formulate the ARIMA model. Estimate the model parameters and utilize the Akaike Information Criterion (AIC) value to identify the best model. The model obtained from the training data set is ARIMA (2,1,1)(1,1,1)₁₂ model, the future one-head monthly cases number of children diarrhea can be obtained.

C. Comparison and discussion

The comparisons of prediction models for the monthly number of children diarrhea are made between the ARIMA model, the single GRNN model, the hybrid EMD-GRNN model and the hybrid EEMD-GRNN model. The actual diarrhea cases number for children and predicted values of different models are illustrated in Fig. 6 and the prediction performances are shown in Table II. Through model comparisons, the proposed hybrid EEMD-GRNN model performs best. It can be observed from Fig. 6 that the predicted values obtained from the proposed EEMD-GRNN model are closer to the actual values than those obtained from the other models. This phenomenon signifies that the hybrid model can combine different advantages from EEMD and GRNN.

As seen from Table II and Fig. 6, it is clear that the hybrid EEMD-GRNN model performs much better than ARIMA model and single GRNN model, and outperforms the hybrid EMD-GRNN model. More precisely, the MAE, RMSE, MAPE and R² of the proposed EEMD-BPN model are, respectively, 664.361, 811.925, 3.1% and 0.991. That these values are smaller than other models. This indicates that there is a smaller

deviation between the actual and predicted values using the proposed EEMD-GRNN model. Thus, the proposed EEMD-GRNN model provides a better prediction result than the other models based on MAE, RMSE, MAPE and R².

The possible reason is that the proposed hybrid model adequately makes use of the advantages of the decomposition methods and GRNN algorithm and integrates them well. In comparisons between EMD-GRNN and EEMD-GRNN, the decomposition method of EEMD is superior to EMD in terms of contribution to the prediction accuracy.

Similarly, the proposed EEMD-GRNN method also performs well in terms of predicting the diarrhea for adult. According to above steps, the ARIMA model generated from the data set is ARIMA (2,1,0)(1,1,1)₁₂. Table III summarizes the diarrhea cases number for adult prediction results using the ARIMA, single GRNN, EMD-GRNN and EEMD-GRNN models. It can also be observed that the proposed EEMD-GRNN model has the smallest MAE, RMSE, MAPE and R² values in comparison with the single GRNN and ARIMA models and hybrid EMD-GRNN model. Thus, the proposed method produces lower prediction errors and outperforms other models with respect to predicting of diarrhea for adult.

TABLE II. PERFORMANCE OF THE FOUR MODELS FOR CHILDREN

Metrics	Models			
	ARIMA	GRNN	EMD-GRNN	EEMD-GRNN
MAE	3364.427	3330.324	2123.084	664.361
RMSE	4437.433	4212.306	3049.842	811.925
MAPE	17.1%	14.9%	14.3%	3.1%
R ²	0.846	0.953	0.9666	0.991

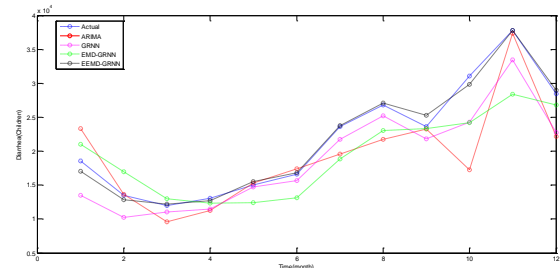


Fig. 6. Prediction results of diarrhea for children.

Fig. 7 depicts the actual diarrhea cases number for adult and the predicted values from the ARIMA, single GRNN, and EMD-GRNN, EEMD-GRNN models. From the Fig. 7, it can be observed that the proposed EEMD-GRNN model provides good prediction results. The predicted values of the proposed model are closer to the actual values than the other three models.

TABLE III. PERFORMANCE OF THE FOUR MODELS FOR ADULT.

Metrics	Models			
	ARIMA	GRNN	EMD-GRNN	EEMD-GRNN
MAE	2999.752	2644.084	1839.788	1110.087
RMSE	3737.192	3643.624	2476.043	1484.657
MAPE	14.09%	11.69%	10.51%	5.14%
R ²	0.823	0.969	0.971	0.995

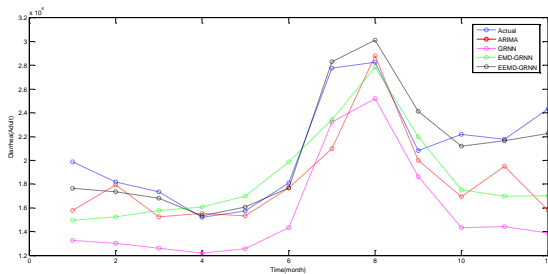


Fig. 7. Prediction results of diarrhea for adult.

V. CONCLUSIONS

In this paper, a novel hybrid approach integrating the EEMD algorithm and the GRNN model is proposed to settle the diarrhea prediction problem. The main contribution of the paper is to propose a novel hybrid method for a stable prediction of nonlinear and non-stationary diarrhea time series data. The proposed method pre-processes the diarrhea time series data and decomposes them into more stationary and regular components (IMFs or residue) using the EEMD technique. Furthermore, the corresponding GRNN model for each divided component is easier to build. After the IMF components and residue are forecasted in the built GRNN model, the prediction values are then aggregated using another independent GRNN model as the final prediction results. This study compared the proposed method with the single GRNN, ARIMA models and hybrid EMD-GRNN model, using MAE, RMSE, MAPE and R^2 as its criteria. Experimental results showed that the proposed EEMD-GRNN model is better and more efficient for prediction diarrhea in Shanghai areas.

There are several advantages of the proposed methodology. First, thanks to the non-linearity and non-stationary of diarrhea, hybrid the EEMD algorithm and GRNN model is a very wise practice for the diarrhea prediction. Moreover, it has rarely been mentioned in previous literature. Thus, applying this hybrid method to forecast diarrhea is very important for the future studies. Furthermore, from the simulation process and results, we can find this hybrid approach is useful in prediction diarrhea. Next, in terms of empirical results, it is a clear finding that the hybrid model can describe them comprehensively. The conventional single prediction models cannot do this very well. However, a hybrid method can integrate the advantages of other single models which conduce to boosting the model prediction ability and enhancing prediction efficiency. From this point of view, in terms of different criteria, it is unsurprising that the hybrid approach performs better than the single ARIMA and GRNN methods, and also superior to other hybrid models, for instance, EMD-GRNN model. Both statistical errors are reduced effectively in this hybrid model. Therefore, the proposed method is very suitable for prediction with nonlinear, non-stationary and strong complexity data, and is an efficient method for diarrhea prediction.

Our study has the some limitations that need further research. First, future studies may aim at combining EEMD and other prediction tools, like support vector regression (SVR), in evaluating the ability of the proposed prediction scheme. Second, integrating GRNN and other time series processing

techniques, such as wavelet transformation and seasonal adjustment method (SAM), in further improving the prediction capabilities can also be investigated in future studies.

ACKNOWLEDGMENT

This research is supported by the Fund of The Shanghai Science and Technology Development Foundation (Grant No. 13430710100). The authors are grateful to the editor and anonymous reviewers for their suggestions in improving the quality of the paper.

REFERENCES

- [1] I. N. Soyiri, D. D. Reidpath, "An overview of health forecasting," *Environmental health and preventive medicine*, vol.18, pp. 1-9, 1998.
- [2] D. Specht, "A general regression neural network," *IEEE Transactions Neural Networks*, vol. 2, pp. 568-576, 1991.
- [3] N. E. Huang, Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, "The empirical mode decomposition and the Hilbert spectrum for nonlinear and nonstationary time series analysis," In: *Proceedings of the royal society of London series a-mathematical physical and engineering sciences*, series A., vol. 454, pp. 903-995, 1998.
- [4] N. E. Huang, Z. Shen, S. R. Long, "A new view of nonlinear water waves: the Hilbert spectrum," *Annu. Rev. Fluid Mech*, vol. 31, pp. 417-457, 1999.
- [5] D. J. Yu, J. S. Cheng, Y. Yang, "Application of EMD method and Hilbert spectrum to the fault diagnosis of roller bearings," *Mech. Syst. Signal Process*, vol. 19, pp. 259-270, 2005.
- [6] Hu, Jianming, Jianzhou Wang, and Guowei Zeng, "A hybrid forecasting approach applied to wind speed time series," *Renewable Energy*, vol. 60, pp. 185-194, 2013.
- [7] D. F. Specht, "A general regression neural network," *IEEE Transactions on Neural Networks*, vol. 2, pp. 568-576, 1991.
- [8] J. D. Wu, J. C. Liu, "A forecasting system for car fuel consumption using a radial basis function neural network," *Expert Systems with Applications*, vol. 39, pp. 1883-1888, 2012.
- [9] W. Yan, "Toward automatic time-series forecasting using neural networks," *Neural Networks and Learning Systems*, *IEEE Transactions on*, vol. 23, pp. 1028-1039, 2012.
- [10] G. Q. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks: The state of the art," *International journal of forecasting*, vol. 14, pp. 35-62, 1998.
- [11] G. J. Bowden, G. C. Dandy, H. R. Maier, "Input determination for neural network models in water resources applications. Part 1- background and methodology," *Journal of Hydrology*, vol. 301, pp. 75-92, 2005.
- [12] Ö. Kisi, "Constructing neural network sediment estimation models using a data-driven algorithm," *Mathematics and Computers in Simulation*, vol. 79, pp. 94-103, 2008.
- [13] K. P. Sudheer, A. K. Gosain, K. S. Ramasastri, "A data-driven algorithm for constructing artificial neural network rainfall-runoff models," *Hydrological Processes*, vol. 16, pp. 1325-1330, 2002.
- [14] P. Flandrin, G. Rilling, P. Goncalves, "Empirical mode decomposition as a filter bank," *IEEE Signal Process. Lett*, vol. 2, pp. 112-114, 2004.
- [15] N. E. Huang, "Review of empirical mode decomposition," *Proc. SPIE*, pp. 71-80, 2001.
- [16] M. C. Wu, C. K. Hu, "Empirical mode decomposition and synchrogram approach to cardiorespiratory synchronization," *Phys. Rev. E*, vol. 73 51917, 2006.
- [17] Z. Wu, N. E. Huang, "Ensemble empirical mode decomposition: a noise-assisted data analysis method, center for ocean-land-atmosphere studies," *Tech Rep*, vol. 51, 2004.

Are We Living in a Happy Country: An Analysis of National Happiness from Machine Learning Perspective

Theresia Ratih Dewi Saputri
Department of Computer Engineering
Ajou University
Suwon, South Korea
trdsaputri@ajou.ac.kr

Seok-Won Lee
Department of Software Convergence Technology
Ajou University
Suwon, South Korea
leesw@ajou.ac.kr

Abstract— National happiness has been actively studied during last ten years. The factor of happiness could be different due to different human perspective. The factors used in this work include both physical needs and the mental needs of humanity such as educational factor. This work identified more than 90 features that can be used to predict the country happiness. Unfortunately, manually analyzing the features is difficult and needs a lot of resources. Due to numerous size of the features, it is unwise to rely on the prediction of national happiness by manual analysis. That process will result in the high cost of analysis. Therefore, this work used machine learning technique which is a Support Vector Machine to learn and predicts the country happiness. Dimensionality reduction is also done in this work. Using the information gain technique, the features can be reduced. This technique is chosen due to its ability to explore the interrelationships among a set of variables. The selected features are also evaluated using the SVM classifier. Using the data of 187 countries from the UN Development Project, this work is able to identify which factor needed to be improved by a certain country to increase the happiness of their citizens.

Keywords-data mining; classification; feature selection; principal component analysis; support vector machine

I. INTRODUCTION

National happiness has been actively studied throughout the last ten years. The work in [8] argues that the government of a country is usually driven by the happiness of their citizens. Some factors that are controlled or authorized by the government positively correlate with the happiness level. That work shows that the key role to determine the citizen happiness is the improvement of public policy. Understanding happiness factors will help governments to make a better policy and legislation.

However, the factors that influence happiness could be different due to different human perspectives. We cannot just simply say that The United State is happier than Indonesia country because The United State has higher GDP. Peggy in [1] stated that happiness is correlated with national economic and

cultural living conditions. The work in [2] determined happiness using three factors which are life expectancy, experienced well-being and Ecological Footprint. Other work in [9] shows a new measurement to improve the happiness of a country. Unlike the previous work, this work studies that happiness is not only related to physical but also mental needs. Therefore, they also consider mental health, which includes stress, depression, and emotional problems.

As a result of the increase of human social complexity, the factors proposed by [2] and [9] may not be reliable anymore. Additional factors such as health and human development index should be examined carefully. However, analyzing the factor to determine happiness of a particular country is not a trivial problem. A single factor can have a bigger impact than another. NEF organization in [2] proposes an equation to calculate the happiness index. However, this equation does not consider the economical aspects. Therefore, this work proposes an approach by extending the factors and adopting machine learning techniques to learn about those factors.

Due to numerous size of the features, it is unwise to rely on the prediction of a national happiness done by manual analysis. That process will result in high cost of analysis. Therefore, this work also proposes the use of machine learning to predict national happiness. Machine learning is a widely known technique to learn about patterns in data. There are several machine learning techniques which can be used to perform a prediction task [3]. One of the remarkable techniques is the support vector machine. This work uses support vector machine because its outstanding ability to perform a classification task.

II. RELATED WORK

This section briefly explains the related work in this project. Firstly, the national happiness analysis is described. It will discuss the importance of happiness analysis. Secondly, the used machine learning is introduced. Lastly, the proposed factor analysis is discussed.

A. National Happiness Analysis

The work in [4] mentioned that happiness could be a good indicator for how well a society is doing. This becomes important because Betham [5] said that the best society is the one where the citizens are happiest. Several researches have been conducted on positive aspects and the matters of happiness in policy making [6][7]. As mentioned in the previous section, happiness can be determined based on various factors. Unfortunately, these factors were analyzed manually [8]. The complexity of the factor leads to the expensive cost of analysis. Therefore, the automatic analysis is needed.

B. Support Vector Machine

Due to its capability to learn from the past experiences, machine learning has been used in various areas. Support vector machine is one of the powerful machine learning algorithm. Support Vector Machine (SVM) is a learning technique which is used for classifying unseen data correctly. It is a learning technique which usually used for classifying the unseen data correctly. This technique has been used in various research field due to its remarkable performance. In order to perform the classification task, support vector machine builds a hyperplane which separates the data into different categories [9].

One of the important advantages of support vector machine is its ability to handle the scarcity of the data. Moreover, support vector machine is able to learn about the complex decision boundaries in the high dimensional feature space efficiently. Due to the complex features used to predict the national happiness, it is important to apply the technique with ability to handle the complex features.

C. Factor Analysis

As mentioned in the previous section, there are a large number of features used to predict the national happiness. However, some of the features may have no significant contribution to the prediction. Therefore, it is unwise to use the entire features to analyze the happiness. This work uses factor analysis to analyze the related features. Factor analysis aims to determine the contribution of a certain feature. This technique does not focus on dimension reduction. Therefore, there will be no features removed. The works in [10] and [11] have introduced the advantages of factor analysis. The first advantages mentioned is the ability to identify latent dimensions or constructs that cannot be done using direct analysis. Moreover, this approach is easy to run and inexpensive in term of resources.

III. PROPOSED APPROACH

The aim of this project is to predict the national happiness of a particular country using machine learning techniques. The proposed approach contains four main steps in the data mining process which are data collection, data preprocessing, data analysis, and classification process as seen in Figure 1.

The first process in the process approach is collecting the data. The data used in this project are gathered from the UN Human Development Project. The data contains of the human development index, GDI, healthy index of each country in the world. However, these data are quite dirty. It cannot be used

directly as the input data for the learning process. Therefore, the second process is data preprocessing. Data preprocessing is used to increase the data quality. By increasing the quality of data results to the increasing number of prediction accuracy and consistency. The processes included in this process are data cleaning and data integration. The routine processes that should be done are filling the missing values, reduce the noise and identify the outliers.



Figure 1. The proposed Approach

The third process in the proposed approach is data analysis. This explanatory data analysis is used for finding the relationship among the attributes of the features. This analysis is done by visualizing the data. Dimensional reduction also be done in this step. Using the information gain technique, the features can be reduced. Information gain technique is used because it can explore the interrelationships among a set of variables. The last part is this work is the classification process. In this classification process, SVM technique is used to predict the happiness of the data based on the important features. The validation process using k-fold cross validation technique is used to measure the performance of the data based on the accuracy, sensitivity and specificity values.

IV. RESULT AND ANALYSIS

This section discusses about the result of each step in the proposed approach. Moreover, this section also presents the analysis of significant factors to determine the happiness of a particular country.

A. Data Collection

As mentioned in the proposed approach section, the data used for this work are gathered from the UN Development Project. In total there are 187 countries listed in the data. Different types of factors are also mentioned in this data, such as human development index, education, environment, health care. The data consists of 105 types of features from 14 different factors.

However, we know that there is no perfect data. This data consists of various missing values, especially for the relatively small country such as Liechtenstein, as seen in Figure 2. This data does not only consist of missing value, but also some of outlier data. Therefore, this work needs a data preprocessing in order to improve the quality of data analysis.

17 Japan	94.8	5.2	..	9.2	3.9	0.0	68.1
18 Liechtenstein	43.7
19 Israel	96.7	4.8	99.7	9.3	3.9	0.3	7.1

Figure 2. Example of Missing value

B. Data Preprocessing

In order to increase the data quality, the data preprocessing is needed. The collected data are scattered in various tables. Therefore, creating an integrated data is needed. The single

integrated table consists of the entire features and sample that we are going to use in the next process. As seen in the previous sub-section, there are some missing values in the data. Those missing values may reduce the quality of the classification process. There are some ways to handle missing values such as pairwise deletion, listwise deletion, and mean substitution.

However, these approaches are not adequate to handle the missing values. The first and second approaches are not adequate because it needs to remove some of the data. Those approaches result in a massive decrease in the sample size for analysis and classification. It may not have a big impact when the number of missing values is small. However, there are a big number of missing values in the data used for this project. Therefore, choosing pairwise or listwise deletion is an unwise decision. One possible approach is to find the substitute of the missing values. Mean substitution can be one of possible solutions. However, adding data with mean will be useless. The overall mean, with or without replacing my missing data, will be the same. Moreover, using mean substitution makes only a trivial change in the correlation coefficient and no change in the regression coefficient.

Therefore, the approach used in this project is regression substitution. Instead of adding a trivial value for the missing value, regression substitution tried to predict the value based on other variables. This technique uses existing variables to make a prediction, and then substitute that predicted value as if it were an actual obtained value. In this case, seven variables are used to predict the missing value.

C. Feature Selection

Selecting the related features is important in order to improve the performance of the classifier. In order to perform this process, WEKA package for attribute selection is used [12]. The evaluator used in this work is information gain. This technique is chosen due to its ability to measure the amount of information in bits about the class prediction [13]. Therefore, it measures the expected reduction in entropy.

NO	Attributes Name	NO	Attributes Name
1	inequality in life expectancy	19	Pupil-teacher ratio
2	homeless person	20	Adolescent birth rate
3	life expectancy at birth	21	Employment to population ratio
4	coefficient of human inequality	22	Gross National Income
5	HDI	23	female suicide rate
6	adult male mortality	24	male suicide rate
7	adult female mortality	25	Sex ratio at birth
8	Total fertility rate	26	Youth unemployment
9	maternal mortality ratio	27	youth literacy rate
10	adult with HIV	28	Electrification rate
11	Orphaned children	29	female secondary education
12	child with HIV	30	Homicide rate
13	under five mortality rate	31	Community
14	Dependency ratio (young age)	32	Internet users
15	Standard of living	33	Birth registration
16	infant mortality rate	34	expected years of schooling
17	Dependency ratio(old age)	35	Health care quality
18	Gender Inequality		

Figure 3. Selected features for classification

There are two main properties in the ranker evaluator. The first property is numToSelect property, which defines the number of attributes to keep, an Integer number that is -1 (all) by default. The next property is the threshold which defines the minimum value that an attribute has to get in the evaluator in order to be kept. In this case, the threshold is set to 0.

After running this process, the number of remaining attributes is 36 includes class attribute. Those attributes are listed in Figure 3. The selected attributes such as inequality in life expectancy (inequality in the distribution of the expected length of life based on data from life tables estimated using the Atkinson inequality index), the number of homeless people, and the mortality rate is used to classify the happiness of a certain country.

D. Classification

In order to evaluate the selected attribute, this work also runs the classification using the entire gathered attributes. The same parameter used to compare both scenarios. The kernel for SVM is chosen based on cross validation. Table 1 shows the result for comparing different types of kernel. We can see that the normalized poly kernel gave an outperform result. Therefore, this kernel is chosen for this work.

TABLE I. COMPARISON RESULT OF THE KERNEL

Kernel Type	Accuracy Rate
Normalize Poly Kernel	68.456 %
Poly Kernel	60.402 %
RBF Kernel	38.926 %
String Kernel	43.624 %

As mentioned before, this work also runs the classification using the entire attributes in order to validate the performance of selected attributes. Using the entire attributes we can see that the classification process result in 58 % of accuracy. This result shows the improvement of accuracy rate and true positive rate. It also shows that using the selected attribute is able to reduce the mean square error. It means that the selected attributes have strong correlation with the class attribute that can be used to predict the class which is the happiness of the country.

E. Analysis

Instead of classifying the data into happy and unhappy countries, this work classified the data into three categories which are happy, mid, and unhappy. Based on the classification results, it shows that most countries in the world are not in a happy state. As seen in Figure 4, 39 %, 38%, 23% of the countries are happy, mid-happy, and unhappy, respectively.

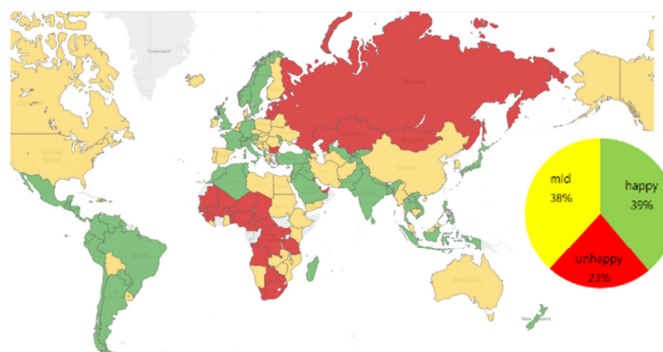


Figure 4. Distribution of Country Happiness

In order to analyze the happiness factor, this work also shows the distribution of the happiness based on the country. A country with red shade is a country in an unhappy state such as

Russia and Nigeria. Moreover, the country with yellow shade is mid-state country such as the United State and Australia. Lastly, the country with green shade has happy state such as Brazill and Indonesia. This figure showed one surprising fact that even though a country is developed, it does not mean that it has a happy state.

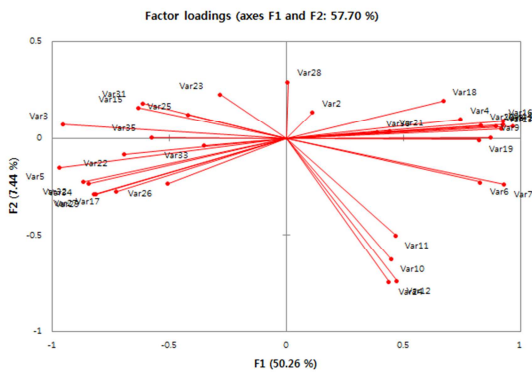


Figure 5. Factor Loadings based on PCA

After the classification process has been done, the next step is factor analysis. This process is done to determine the factor that drives the happiness of a country. There are various approaches in statistical learning to analysis the factor. This work uses factor analysis with principal component analysis (PCA) to evaluate the factors. Using this approach, the selected features are grouped into several factors as seen in Figure 5. The evaluation of the factors is done based on the classification results. In order to evaluate the factors, each of the sample was observed based on its class attributes. Using this evaluation, we can clearly see which features make a significant contribution to determine the happiness of a particular country.

• Increase to improve happiness	• Decrease to improve happiness
<ul style="list-style-type: none"> ✓ adult female mortality ✓ adult male mortality ✓ Health care quality ✓ Employment to population ratio ✓ Youth unemployment ✓ Total fertility rate 	<ul style="list-style-type: none"> ✓ Inequality in life expectancy ✓ homeless person ✓ adult with HIV ✓ Orphaned children ✓ Dependency ratio(young age) ✓ Gender Inequality ✓ Standard of living ✓ Internet users ✓ female suicide rate ✓ male suicide rate

Figure 6. Summary of Significant Factors

In order to determine the significant factors, the factor loading values are used. Based on those values, some features should be increased to improve the happiness of a certain country such as health care quality and employment population ratio. We can also determine which features should be decreased in order to improve the happiness such as inequality in life expectancy, homeless person, adults with HIV and gender inequality. Figure 6 shows a summary of the features that need to be decreased or increased based on the results of the factors analysis process. In order to improve the state of happiness in for a country, the government can refer to the listed significant factors in the policy making process. Using this list, it will be easier for them to determine which factors

they need to improve in order to increase the happiness of their citizens.

V. CONCLUSION

The happiness of a country cannot simply determine by its development index. This work showed that there are various factors can be used to determine the happiness. Due to the various factors to classify the happiness, prediction is hard to perform. Therefore, this work proposed the use of machine learning technique to learn about the factor to predict the national happiness combined with feature selection approach.

This work showed that the feature selection process using information gain is able to increase the performance of the classification process. The performance improvement is proved by the increase of accuracy rate when the selected features were used. This work also shows that using SVM classifier the happiness of each country can be determined effectively and efficiently.

Even though the accuracy rate increased due to the use of selected features, unfortunately, the accuracy of the result is still inferior. We argue that the inadequate of accuracy is caused by the large number of missing values. Therefore, the future work should implement a reliable approach for handling the missing values.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2013R1A1A2009801)

REFERENCES

- [1] Schyns, Peggy. "Crossnational differences in happiness: Economic and cultural factors explored." *Social Indicators Research* 43.1-2, pp. 3-26, 1998.
- [2] <http://www.happyplanetindex.org/assets/happy-planet-index-report.pdf>
- [3] Witten, Ian H., and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
- [4] Gudmundsdottir, Dora Gudrun. "The impact of economic crisis on happiness." *Social indicators research* 110.3. pp: 1083-1101, 2013.
- [5] Bentham, J. (1789/1996). *An Introduction of the principles of morals and legislation*. Oxford: Clarendon Press. (Originally from 1789)
- [6] Diener, E., Lucas, R. E., Schimmack, U., & Helliwell, J. *Well-being for public policy*. New York: Oxford University Press, 2009.
- [7] Dolan, Paul, and Mathew P. White. "How can measures of subjective well-being be used to inform public policy?." *Perspectives on Psychological Science* 2, no. 1 pp.71-85.2007.
- [8] Viinamäki, H., Kontula, O., Niskanen, L., & Koskela, K. "The association between economic and social factors and mental health in Finland." *Acta Psychiatrica Scandinavica* 92, no. 3, pp.208-213, 1995.
- [9] Malhotra, R., & Jain, A. "Software Effort Prediction using Statistical and Machine Learning Methods." *International Journal of Advanced Computer Science and Applications* 2., pp. 1451-1521, 2011.
- [10] Garson, G. David, "Factor Analysis," from *Statnotes: Topics in Multivariate Analysis*.
- [11] Tucker, L. R., & MacCallum, R. C.. "Exploratory factor analysis." Unpublished manuscript, Ohio State University, Columbus, 1997.
- [12] <http://weka.sourceforge.net/doc.dev/weka/filters/supervised/attribute/AttributeSelection.html>
- [13] Roobaert, D., Karakoulas, G., & Chawla, N. V. "Information gain, correlation and support vector machines." In *Feature Extraction*, pp. 463-470. Springer Berlin Heidelberg, 2006.

BiBinConv_{mean}: A Novel Biclustering Algorithm for Binary Microarray Data

Haifa BEN SABER

Mourad ELLOUMI

Time Université

Laboratory of Technologies of Information and
Communication and Electrical Engineering (LaTICE)
National High School of Engineers of Tunis (ENSIT),
University of Tunis, Tunisia.

Laboratory of Technologies of
Information and Communication and
Electrical Engineering (LaTICE)
University of Tunis-El Manar, Tunisia.

Abstract— In this paper, we present a new algorithm called, BiBinConv_{mean}, for biclustering of binary microarray data. It is a novel alternative to extract biclusters from sparse binary datasets. Our algorithm is based on Iterative Row and Column Clustering Combination (IRCCC) and Divide and Conquer (DC) approaches, K-means initialization and the CroBin evaluation function [6]. Applied on binary synthetic datasets, our algorithm outperforms other biclustering algorithms for binary microarray data. Biclusters with different numbers of rows and columns can be detected, varying from many rows to few columns and few rows to many columns. Our algorithm allows the user to guide the search towards biclusters of specific dimensions.

Keywords— component; Biclustering, binary data, microarray data, Iteratif Row Column Combinaison approach, Divide and Conquer approach, CroBin.

I. INTRODUCTION

A DNA Microarray is a glass slide covered with a product and DNA samples containing thousands of genes [8]. Biclustering of microarray data can be helpful to study, among others, the activity and the condition of the tissue via microarrays such as transcription factor binding, insertional mutagenesis and gene expression data. It can be helpful also to find genes involved in tumor progression, identify the function of new genes, rank the tumors into homogenous groups and identify new therapeutic strategies.

Biclustering algorithms of binary microarray data enable to extract useful biclusters from binary data to provide information about the distribution of patterns and intrinsic correlations [16][15]. A number of biclustering algorithms of binary microarray data have been proposed in recent years, such as the Biclustering Bit-pattern (BiBit) [14][1], Cmnk [11], [9], BiMax [13], Bipartite Bron-Kerbosch (BBK) [12], Binary Matrix Factorization (BMF) [18], e-CCC Biclustering [5], e-BiMotif [5], BIMODULE[4], BIDENSE[4], CETree [4], DeBi [17] and Maximal Frequent Item Set [17]. Besides, there are also other approaches based on Gaussian or Latent Mixture Models, BEM and BEM [3].

In the same context, different biclustering algorithms have been adapted to deal with biclustering of binary gene expression data. However, these changes lead to more complicated user input parameters. Besides, all the elements of every generated bicluster are set to zero in the input matrix, introducing noise.

It is an interesting from the biological point of view [12] to search biclusters with small proportion of zeros especially when binary data matrix is obtained after normalization and binarization. However, most of biclustering algorithms of binary microarray data, including Cmnk and BiMax, fail to extract pertinent biclusters on sparse binary datasets. Indeed, if we apply one of these algorithms on a typical sparse binary microarray datasets (with thousands of columns), most of the extracted biclusters are made up only by 1's.

The rest of this paper is organized as follows. In section 2, we introduce some preliminaries. In section 3, we present BiBinConv_{mean} our biclustering algorithm of binary microarray data. In section 4, we present an illustrative example. In section 5, we present the experimental results obtained thanks to BiBinConv_{mean} on binary synthetic, and we compare these results with those obtained by other biclustering algorithms of binary microarray data. Finally, in section 6 we present the conclusion.

II. PRELIMINARIES

In this section we present some preliminaries necessary for the presentation of important formulas and relationships and the used theory.

Let $I = \{1, 2, \dots, n\}$ be a set of indices of n genes, $J = \{1, 2, \dots, m\}$ be a set of indices of m conditions and $M(I, J)$ be a data matrix associated with I and J .

The biclustering problem of a binary microarray data boils down to a minimization of the criterion $W(z, w, A)$ defined by:

$$W(z, w, A) = \sum_{k=1}^a \sum_{l=1}^m \sum_{i \in z_k} \sum_{j \in w_l} |m_{ij} - a_{kl}|. \quad (1)$$

where:

- z is a partition of I into g clusters.
- w is a partition of J into m clusters.
- A is the summary of the data matrix where k (resp. l) represents number of clusters on rows (resp. columns). We note that the bicluster kl is defined by the m_{ij} with $z_{ik}w_{jl} = 1$.

III. CONTRIBUTION

In this section, we develop our biclustering algorithm, BiBinConvmax that is based on the IRCCC approach, K-means initialisation and the CroBin evaluation function. It consists to permute the rows and the columns in order to obtain homogeneous biclusters. As a preprocessing step of this algorithm is applied:

a) First, when the data matrix M is not a binary one, we apply a thresholding function to transform it to a binary one. To the best of our knowledge, the main thresholding functions are discretize, normalize and binarize [7]. According to [14], [9][14], [9], the most adequate thresholding function to binarize microarray data is binarize.

b) Then, we make an initial clustering z^0 of rows and an initial clustering w^0 of columns, thanks to k-means algorithm [10] [2].

After the initialization, we update the clustering of rows and columns.

The preprocessing step consists in obtaining a binary matrix M_b that can be directly used by our algorithm. The binary values of 1 and 0 under an experimental condition c mean that a gene is expressed or not, respectively. For example, in [13], a discretization threshold was set to $e^2 + (e^1 - e^2) = 2$, with e^1 and e^2 as the minimum and maximum expression values in the data matrix, respectively.

Our biclustering algorithm, BiBinConv_{mean} receives as input a binary matrix M gives as output $(z^{opt}, w^{opt}, A^{opt})$, where z^{opt} is the final clustering of rows of M_b , w^{opt} is the final clustering of columns of M_b and A^{opt} is the summary matrix related to z^{opt} and w^{opt} .

By adopting BiBinConv_{mean}, we operate as follows:

First, we compute (z^0, w^0, A^0) thanks to k-means algorithm [10], [2], where z^0 is the initial clustering of rows of M_b , w^0 is the initial clustering of columns of M_b and A^0 is the summary matrix related to z^0 and w^0 .

Then, we repeat this process :

We compute (z^c, w^{c-1}, A^c) starting from $(z^{c-1}, w^{c-1}, A^{c-1})$, where A^c is an intermediate summary matrix

We compute (z^c, w^c, A^c) starting from (z^{c-1}, w^{c-1}, A^c)

Until $(z^c, w^c, A^c) = (z^{c-1}, w^{c-1}, A^{c-1})$.

IV. ILLUSTRATIVE EXAMPLE

We present in this section steps to perform the biclustering on binary datasets. Our algorithm allows to reorder the rows and the columns of the data matrix in both dimensions to obtain homogeneous biclusters. The algorithm minimizes the difference between the initial matrix according the two way and the ideal matrix.

To illustrate our algorithm method, we propose to run it on an simple example. Let M_b be a (4,5) matrix of binary data to perform biclustering. The initialization is to group the rows and columns with K-means reference algorithm method. After initialization, we compute z and w matrix whose elements determine the membership of rows or column in horizontal or vertical clusters, respectively.

Then, we reorganize the binary matrix. After that, we compute the summary matrix is obtained from z and w and the bicluster kl is defined by the x_{ij} 's with $z_{ik}w_{jl} = 1$.

The summary matrix is presented by the major value a_{kl} which presents the degree of homogeneity via summary matrix. We update clusters on rows and columns by computing our criterion on both dimensions. Initial binary matrix M_{bis} given by :

	C1	C2	C3	C4	C5
G1	1	1	0	1	0
G2	0	0	1	0	1
G3	1	1	0	1	0
G4	0	0	1	0	1

- **Initialization:**

$z^0 = (1, 2, 2, 3), w^0 = (1, 1, 0, 0, 0), A^0 = (1, 0, 1, 1, 0, 1)$

- **Iteration 1:**

c = 1

We compute (z^1, w^0, A^1) starting from (z^0, w^0, A^0) , we obtain:

$(z^1, w^0, A^1) = ((1, 3, 2, 1); (1, 1, 0, 0, 0); (1, 1, 1, 0, 0, 1))$

We compute (z^1, w^1, A^1) starting from (z^1, w^0, A^1) , we obtain:

$(z^1, w^1, A^1) = ((1, 3, 2, 1); (2, 2, 1, 2, 1); (1, 1, 1, 0, 0, 1))$

We obtain $(z^2, w^2, A^2) \neq (z^1, w^1, A^1)$.

- **Iteration 2: c = 2**

We compute (z^2, w^1, A^2) starting from (z^1, w^1, A^1) , we obtain

$(z^2, w^1, A^2) = ((2, 1, 2, 3); (2, 2, 1, 2, 1); (0, 1, 1, 0, 0, 1))$

We compute (z^2, w^2, A^2) starting from (z^2, w^1, A^2) , we obtain:

$(z^2, w^2, A^2) = ((2, 1, 2, 3); (2, 2, 1, 2, 1); (0, 1, 1, 0, 0, 1))$

We obtain $(z^3, w^3, A^3) \neq (z^2, w^2, A^2)$

- **Iteration 3: c = 3**

We compute (z^3, w^2, A') starting from (z^2, w^2, A^2) , we obtain:

$$(z^3, w^2, A') = ((1, 1, 1, 1); (2, 2, 1, 2, 1); (1, 0))$$

We compute (z^3, w^3, A') starting from (z^3, w^2, A') , we obtain:

$$(z^3, w^3, A') = ((2, 1, 2, 3); (2, 2, 1, 2, 1); (1, 0))$$

We obtain $(z^3, w^3, A') \neq (z^2, w^2, A^2)$.

- **Iteration 4: c = 4**

We compute (z^4, w^3, A') starting from (z^3, w^3, A') , we obtain:

$$(z^4, w^3, A') = ((1, 2, 1, 2); (2, 2, 1, 2, 1); (1, 0, 0, 1))$$

We compute (z^4, w^4, A') starting from $(z^4, w^3, A') = (z^3, w^3, A')$, we obtain:

$$(z^4, w^4, A') = ((1, 2, 1, 2); (1, 1, 1, 1, 1); (0, 1))$$

We obtain $(z^4, w^4, A') \neq (z^3, w^3, A^3)$.

- **Iteration 5: c = 5**

We compute (z^5, w^4, A') starting from (z^4, w^4, A') , we obtain:

$$(z^5, w^4, A') = ((1, 2, 1, 2); (1, 1, 1, 1, 1); (0, 1))$$

We compute (z^5, w^5, A') starting from (z^5, w^4, A') , we obtain:

$$(z^5, w^5, A') = ((1, 2, 1, 2); (1, 1, 2, 1, 2); (1, 0, 0, 1))$$

We obtain $(z^5, w^5, A^5) = (z^4, w^4, A^4)$.

After five iterations, we obtain $(z^{opt}, w^{opt}, A^{opt}) = (z^5, w^5, A^5)$.

	C1	C2	C4	C3	C5	Z
G1	1	1	1	0	0	1
G3	1	1	1	0	0	1
G2	0	0	0	1	1	2
G4	0	0	0	1	1	2
w	1	1	1	2	2	X

Colored blocs represent the obtained biclusters thanks to BiBinConv_{mean}

IV. EXPERIMENTAL RESULTS

We have experimented the BiBinConv_{mean} algorithm on simulated data. To generate our simulated data, we operate as follows:

We choose the number of biclusters; in our case we choose 3 clusters on rows ($g = 3$) and 2 clusters on columns ($m = 2$).

We use Latent Bernoulli Mixture (LBM) model to generate mixtures by considering patterns of overlapping well separated +5% or fairly separated: ++ 15% or poorly Separated +++25% and sizes of data are used; small one (50, 30), medium one (100, 60) and large one (200, 120).

For instance, to apply the simulation on (100, 60) matrix for 10 samples with a low degree of mixing such that a 3

clusters proportions on rows p_k , 2 clusters proportions on columns q_l

Where:

$p_k = [0;2; 0;3; 0;5]$, $theta_0 = [0;3; 0;7]$, $Alpha_0 = [0;7; 0;3; 0;3; 0;7; 0;7; 0;7]$ and a degree of mixture belonging $[0;0; 0;05]$.

The BiBinConv_{mean} algorithm is fast and gives good results when the biclusters have the same proportions and degrees of homogeneity similar except that you set the number of clusters on the rows and columns. However, it seems to be bad when the proportions of partitions are dramatically different which leads to think that BiBinConv_{mean} assumes equal proportions of clusters. This algorithm rearranges the rows and columns of the data matrix along the two sheets of the rows and columns of homogeneous biclusters. The algorithm minimizes the difference between the initial matrix structured and the ideal matrix according scores.

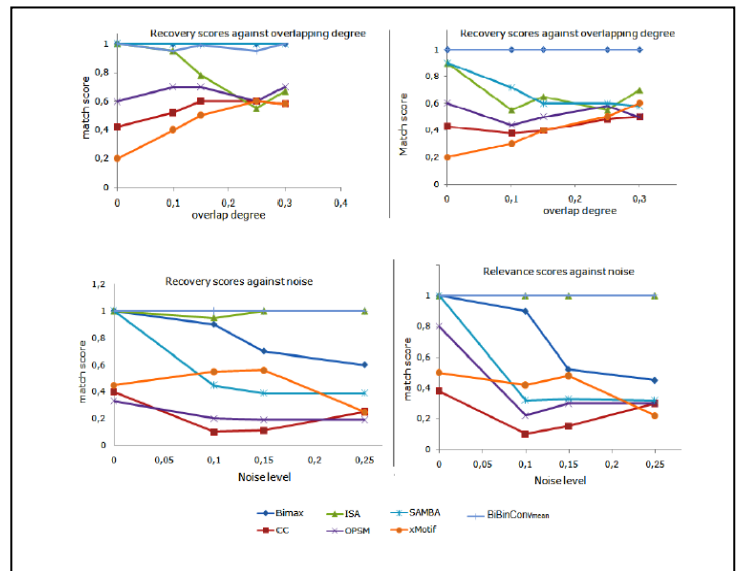


Figure 1. Recovery Scores against overlapping degree and noise

The summarize of the most important points obtained from these simulations are as follows: Obviously, we can interpret the reasonable results according to the model underlying the data structure. In this research, we proves that if the proportions of the components are considered equal give good results. The convergence has a fast progress: most of the time. The BiBinConv_{mean}, is faster and gives us interesting results.

The obtained bicluster is very clear since extracted biclusters containing a majority number of '1' and very illustrative to help the biologist to extract knowledge. According to our implementations, we note first that the choice and application of a given criterion is not always obvious or easy to find. According to our study, we find that BiBinConv_{mean} does not give good results when the matrix

becomes more large. We remark also that the error rates are proportional according to the overlap rate.

V. CONCLUSION

In this paper, we presented our method of binary biclustering considered to be relevant by the expert starting from the data resulting from the microarrays. The suggested method generates interesting biclusters. To achieve this purpose, we selected data to which we applied a thresholding to release the binary data. Our proposal has been implemented and evaluated on synthetic datasets.

According to our implementation, we note first that the choice and application of a given criterion is not always obvious or easy to find. Enjoying the benefits of our algorithm, we proposed a methodology for the identification of homogeneous biclusters. The first results are very encouraging and persuade us of the obvious interest of such an approach.

Finally, further analysis and biological validation of the obtained results is under study.

REFERENCES

- [1] Aguilar-Ruiz and Jesús S. Shifting and scaling patterns from gene expression data. *Bioinformatics*, 21(20):3840–3845, 2005.
- [2] Khalid Benabdeslem and Kais Allab. Bi-clustering continuous data with self-organizing map. *Neural Computing and Applications*, 22(7):1551–1562, 2013.
- [3] M. Charrad. Une approche genrique pour l-analyse croisant contenu et usage des sites web par des methodes de bipartitionnement. PhD thesis, Paris and ENSI, University of Manouba, 2010.
- [4] Jiun-Rung Chen and Ye-In Chang. A conditionenumeration tree method for mining biclusters from dna microarray data sets. *Elsevier*, 97:44–59, 2007.
- [5] Joana P. Goncalves and Sara C. Madeira. e-bimotif: Combining sequence alignment and biclustering to unravel structured motifs. In *IWPACBB*, volume 74, pages 181–191, 2010.
- [6] Gerard GOVAERT. La classification croisee. *Modulad*, 1983.
- [7] Santamaria R. Khamiakova T. Sill M. Theron R. Quintales L. Kaiser, S. and F. Leisch. *biclust: Bicluster algorithms. R package.*, 2011.
- [8] Ouafae Kaissi. *Analyse de Données Transcriptomiques pour La Recherche de Biomarqueurs Liés à Certaines Pathologies Cancéreuses.* PhD thesis, University Abdelmalek Essaadi, Tangier, Morocco., sep 2014.
- [9] Mehmet Koyuturk. Using protein interaction networks to understand complex diseases. *Computer*, 45(3):31–38, 2012.
- [10] G. C. Marcos A.S. da Silva AND, Antonio M.V. Monteiro AND. Somcode: Design patterns and generic programming in the implementation of self organizing maps. *BMC Genomics.*, 2013.
- [11] Ananth Grama Mehmet Koyuturk, Wojciech Szpankowski. Biclustering gene-feature matrices for statistically significant dense patterns. In *2004 IEEE Computational Systems Bioinformatics Conference (CSB'04)*, pages 480–484, 2004.
- [12] Stefan Bleuler Oliver Voggenreiter and Wilhelm Gruissem. Exact biclustering algorithm for the analysis of large gene expression data sets. *Eighth International Society for Computational Biology (ISCB) Student Council Symposium Long Beach, CA, USA.*July, pages 13–14, 2012.
- [13] Amela Prelic, Stefan Bleuler, Philip Zimmermann, Anja Wille, Peter Bühlmann, Wilhelm Gruissem, Lars Hennig,
- [14] Lothar Thiele, and Eckart Zitzler. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22:1122–1129, 2006.
- [15] Perez-Pulido A. J. Rodriguez-Baena, D. S. and J.S. Aguilera-Ruiz. A biclustering algorithm for extracting bit-patterns from binary datasets. *Bioinformatics.*, 2011.
- [16] Bhattacharyya D. K. Roy, S. and J. K. Kalita. Cobi: Pattern based coregulated biclustering of gene expression data. *Pattern Recognition Letters.*, 2013.
- [17] Akdes Serin. *Biclustering analysis for large scale data.* Phd., 2011.
- [18] Akdes Serin and Martin Vingron. Debi: Discovering differentially expressed biclusters using a frequent itemset approach. *Algorithms for Molecular Biology*, 6:18, 2011.
- [19] Chris Ding Xian Wen Ren Xiang Sun Zhang Zhong Yuan Zhang, Tao Li. Binary matrix factorization for analyzing gene expression data. *Data Mining and Knowledge Discovery*, 20:28–52, 2010.
- //
- [20] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” *Phil. Trans. Roy. Soc. London*, vol. A247, pp. 529–551, April 1955.

Integration testing criteria for mobile robotic systems

Maria A. S. Brito*, Marcos P. Santos*, Paulo S. L. de Souza* and Simone do R. S. de Souza*

*Department of Computer Systems

University of São Paulo, São Carlos, SP, Brazil

Email: masbrit,mpereira,pssouza,srocio@icmc.usp.br

Abstract—Testing activity applied to mobile robotic systems is a challenge because new features, such as non-determinism of inputs, communication among components and time constraints must be considered. Simulation has been used to support the development and validation of these systems. Coverage testing criteria can contribute to this scenario adding mechanisms for measuring quality during the development of systems. This paper presents a test model and a set of coverage criteria to test the interaction among the components of mobile robotic systems. The model and criteria focus on robotic systems developed in ROS, a Robotic Operational System in which communication is established through publish/subscribe interaction schema. The testing criteria were evaluated using a robotic application. The results confirm that the use of coverage testing criteria has advantages for integration testing of mobile robotic systems.

I. INTRODUCTION

Research on software methods and environments of robotics development has increased over the past decade. The development of robotic systems requires a middleware to provide tools for interfacing with different system modules, hardware abstraction and communication facilities. Many robotic systems are custom-designed for specific projects and involve high costs of software and hardware. Some middlewares used to develop applications for robotics and complex systems exhibit different behaviors and specificities, as well as unique qualities that make them better suited to a particular task [9]. Various robot middlewares have been proposed, such as [11] Player [10], Orocos [5], Orca [4] and ROS [18].

The validation of a mobile robotic application normally includes simulations and software testing techniques. VSTs (Virtual Simulation Tools/Technology) have been widely used in development environments to model, simulate and evaluate different aspects of a system before they are implemented in an objective platform [9]. They reduce the development time because all tasks are grouped into the same environment. The validation of these systems is a challenging task due to the amount of human resources, equipment and technical support required for the production of reliable results and safety assurance. Some tasks may be highly risky and uncertain, especially those that involve transferring results from an offline simulation to the real world [6].

Mobile robots are normally distributed computing systems because they involve many heterogeneous components, multiple computers and devices. A common communication mechanism of such systems is the publish/subscribe interaction schema, which provides a loosely coupled form of interaction

in large scale settings and comprises subscribe and publish nodes. Subscribers express their interest in an event, or a pattern of events, and are notified of any event generated by a publisher that has matched their registered interest [8]. Multiple concurrent publishers and subscribers may be connected for a single topic and one node may publish and/or subscribe to multiple topics. Nodes are processes that perform computation. These common features of the publish/subscribe (e.g., event correlation, communication channels, timestamp aspects) require more specific testing approaches, which can show defects in such critical applications. Integration testing activity is an emerging and promising research direction, but it still lacks testing criteria to reveal faults in this domain.

In this direction Kang et al. [13] proposed a simulation-based interface testing automation tool (SITAT) for robot software components which generates and executes test cases in a simulation environment. A similar approach is presented in [12], which proposes a required and provided interface specification-based testing method (RPIST). Both approaches require the specification of the application interfaces. Lim et al. [15] defined a hierarchical testing model and its testing automation framework for robot technology component (RTC). Integration testing is based on the interoperability of hardware and software components. A built-in unit/integration test framework called ROStest was defined to support software testing of the ROS-based systems [2]. ROStest is an integration test suite compatible with xUnit frameworks. Its main disadvantages are the need of writing test code and changes in the source code require changes in the test code.

In concurrent programming, several processes of an application communicate among themselves to solve a problem. A process is a program in execution formed by an executable program, its data, a program counter, other registers and all information required for its execution [22]. This communication feature of the concurrent program resembles the behavior of a mobile robotic application in which several nodes interact to solve a computation. Some approaches of testing integration on concurrent programs explore communication and synchronization among processes [3], [7], [20], [21]. We have revisited these approaches to check the similarities in the testing applied and related to the mobile robotic applications that use the publish/subscribe interaction and, therefore the approach proposed here is inspired on the testing criteria for concurrent programs [19].

This paper proposes an approach of integration software testing for robotic systems to improve the quality of these systems. The robotic systems considered are composed of a set of distributed nodes that communicates by message passing using publish/subscribe schema. Therefore, our approach is

concerned with the communication among software components implemented in ROS system.

The paper is organized as follows: Section II introduces concepts of publish/subscribe schema and integration testing; Section III provides a motivating example for the problem characterization; Section IV describes our testing approach and the proposed testing criteria in detail; Section V illustrates the application of the approach in a case study; Section VI presents the related work and, finally, Section VII draws conclusions and recommends future work.

II. BACKGROUND

This section provides some concepts of the publish/subscribe interaction schema and the integration testing addressed.

A. Publish/Subscribe interaction schema

The publisher/subscribe interaction schema studied is implemented in ROS (Robotic Operational System) [18]. ROS is a meta-operating system that consists of an open-source library and tools provided by Willow Garage for the development of robotic applications [18]. It is integrated with tools and libraries, such as OpenCV, a library of programming functions for computer vision of real time, Point Cloud Library (or PCL) for point cloud processing, Gazebo, a multi-robot simulator for outdoor environments, and Player, a network server for the robot control that provides a clean and simple interface to the robots sensors and actuators over the IP network.

A distributed system developed in ROS is formed by many nodes that either run in a single machine or are distributed over different machines. The communication among nodes can be established through two basic mechanisms (Figure 1). The first uses services (synchronous communication) that enable nodes to send or receive requests to/from another node. The second is the publisher/subscribe interaction (asynchronous), in which a node can publish messages in a topic, as well as subscribe from a topic to receive messages from other nodes. Our test model focuses on the publish/subscribe interaction [8].

A node is an executable file that uses ROS to communicate with other nodes by sending messages. A message is a data structure that comprises typed fields. A node sends out a message by publishing it to a given topic. The topic is a name used for the identification of the content of a message. If a subscribe node expresses its interest in an event, or a pattern of events, it is notified of any event generated by a publisher that has matched its registered interest. An event is asynchronously propagated to all subscribers that have registered interest in it.

The publish/subscribe interaction provides the loosely coupled interaction required in such large scale settings. This loosely coupling occurs because of three features: referential decoupling, flow decoupling and time decoupling [8]. When two nodes communicate, they need to know only the type of data they will produce or consume; no extra information is required. This feature is called referential decoupling. Flow decoupling occurs when two processes do not block each other when a message has been sent. Time decoupling enables the transmission or reception of a message at any time. Section III provides an example to illustrate a system described in ROS.

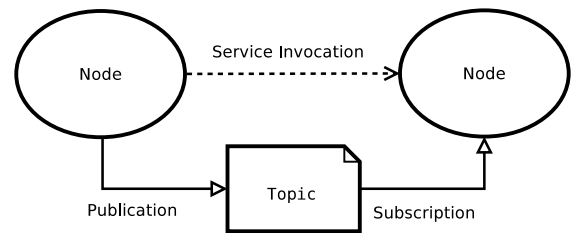


Fig. 1. ROS Communication Model [1].

B. Integration Testing

Different connotations are used for the test phases. We consider the test of the communication among methods of different components (one or more classes) that compose a system an integration testing.

Integration testing should be conducted after the unit testing and the most used test case designs explore the inputs and outputs of data, despite the techniques that exercise specific paths of the source code of the program [17]. This design form is more common because the objective is to exercise the interactions and not the features of units. The integration testing focuses on definitions and uses of variables in different units responsible for the communication. Three types of integration errors can be found [14]: 1) Interpretation error, which occurs when a unit has implemented a functionality different from the specification. Examples include wrong, extra or missing functions; 2) Miscoded call error, which occurs when the developer has inserted a call instruction in the wrong point of the program. Examples are a call instruction on a path or a statement which should not have the call; and 3) Interface error, which occurs whenever the interface standard between two modules has been violated. Examples include incorrect parameters, data types, format and input/output modes. This work explores mainly the miscoded call and interface errors.

III. MOTIVATING EXAMPLE

This section addresses the identification of an error that may occur in the communication among components. Figure 2 shows an example of a layout of a robotic application developed using ROS. This application is a simplified version of iRobot Roomba, which explores unknown environments to clean floors. It contains five components, called processes in this paper, namely *Controller*, *Proximity Sensor*, *Collision Avoidance*, *Motor Driver* and *Mapping*, illustrated in the graph generated from the ROS. The communication among these processes is established using the following topics: *odometry*, *proximity*, *cmd_motors*, *velocity*, *location*, *time_To_Impact* and *mapper_activation*.

Listings 1 and 2 show two excerpts of the codes related to the *Collision Avoidance* [16] and *Controller* processes. The *Controller* process receives information from *time_to_impact Callback* topic (line 11) published by the *Collision Avoidance* process and processes it. In the next step, it publishes data in the *cmd_motors* topic (line 29), which will be subscribed by the *Motor Driver* process. The *Controller* process also publishes the robot speed in the *velocity* topic (line 28), which

will be subscribed by the *Collision Avoidance* process (line 8). Each code has two *callback* functions that contain a code related to the incoming messages. ROS will call the *callback* function once for each arriving message. For example, if the *Controller* process has published data in the *velocity* topic, the *callback* function, called *velocityCallback*, is activated (line 8). The *Collision Avoidance* process will be notified that a process has published data of its interest, because it has subscribed in the *velocity* topic. The *time_to_impactCallback* function is invoked when data have been published in the *time_To_Impact* topic (line 30).

```

1 //Collision Avoidance Process
2 State State; // {working, slowed, stopped,
   crashed}
3 int proximity;
4
5 proximityCallback(int proximityP) {
6     proximity = proximityP;
7
8 velocityCallback(int speed) {
9     if (speed == 0)
10        time_to_impact = 9999;
11    else
12        time_to_impact = proximity /
13        speed;
14    if (time_to_impact < 2) {
15        if (state != working)
16            state = stopped;
17        else {
18            state = crashed;
19            assert(false); //
20            Failure
21        }
22    }
23    else
24        if (time_to_impact < 3) {
25            state = slowed;
26            reduceMapping();
27        }
28        else {
29            state = working;
30            activeMapping();
31        }
32    publish(time_to_impact);
33 }

```

Listing 1. Excerpt of the *Collision Avoidance* [16] process.

```

1 //Controller Process
2 State state; //{working, slowed, stopped,
   crashed}
3 int time_to_impact;
4 int speed; //{0, 1, 2}
5 int c_motors; //{stop, reduce, working}
6
7 odometryCallback(int odometryX) {
8     odometry = odometryX;
9 }
10
11 time_to_impactCallback(int timeToImpactT) {
12     time_to_impact = timeToImpactT;
13     if (time_to_impact < 2) {
14         c_motors = stop;
15         speed = 0; //break
16         state = crashed;
17         activeMapping();
18     } else if (time_to_impact <= 3) {

```

```

   c_motors = reduce;
   speed = 1;
   reduceMapping();
   state = slowed;
} else {
   c_motors = working;
   speed = 2;
   state = working;
}
publish(speed);
publish(c_motors);

```

Listing 2. Excerpt of the *Controller* Process

Communication problems may occur when the interaction among processes is intense. An example of a defect that can be identified is the different frequency of publication from two processes communicating. Line 30 specifies the publication of the data in the *timeToImpact* topic. If, for any reason, the data have been published faster than the *Collision Avoidance* process can access, some data can be lost. If that happened, after their processing the *Collision Avoidance* process would produce incorrect outputs and the system would exhibit a non-expected behavior.

IV. INTEGRATION TESTING APPROACH

Our integration testing approach aims at revealing defects related to the communication among processes of a robotic application. The defined testing criteria exercise the input and output data of the processes, frequency of publishing of the messages and failures of an application. The testing approach uses the source code of a program as input to derive the elements required for each criterion based on a graph. The next step is to create the inputs for the application or the sets of test cases able to cover the required elements. The application is executed and the coverage is analysed for each criterion. New test data can be generated to improve the coverage of the elements required until the maximum coverage has been obtained.

A test model was defined to capture the communication interfaces and data flow among processes. This model is based on the work of Souza et al. [19], [21], who defined the structural testing criteria for MPI (Message Passing Interface) exploring interactions among processes of a concurrent application. Our test model extends this work analyzing specific details of publish/subscribe communication as the loosely coupled among processes, and non-determinism during the processing of the *callbacks* (from threads and queues).

The proposed test model represent the application using a composition of two types of graphs, the graph generated by the ROS and Control Flow Graphs (*CFGs*). A fixed number of processes np is created at the beginning of the application. In our model, p is a process that performs computations and communicates with another process using streaming topics, RPC services, and the parameter server of the ROS. Set of processes P and inter-processes edges T are represented in the Publish/Subscribe-based Def-Use graph (*PSDU*). Each process p of the *PSDU* graph has its own internal structure which is represented by a *CFG*.

A *CFG* is a directed graph that represents the structure of a program as nodes and edges. Nodes in the *CFG* represent

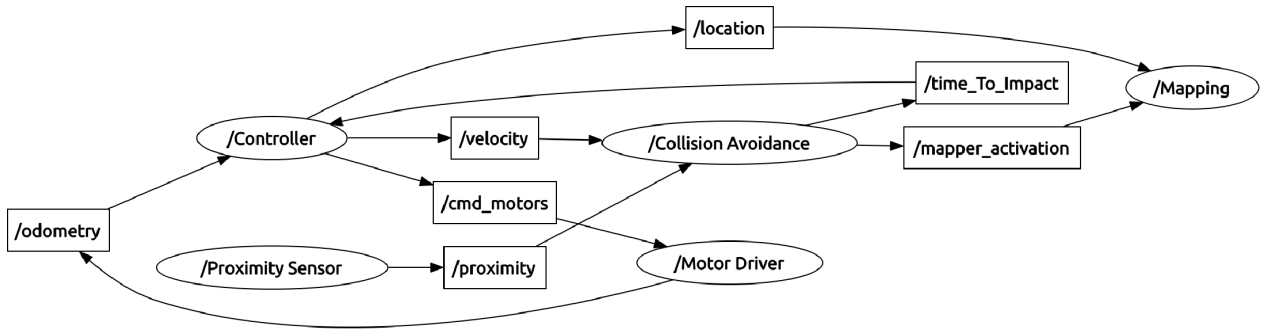


Fig. 2. ROS Graph of the iRobot System.

blocks of sequential statements such as if any one statement of the block is executed, then all statements in the block are executed. The edges represent the communication among the nodes of a *CFG*.

A definition (*def*) is a location in the program where a value for a variable is stored into memory. A use occurs in a location where a value of variable is accessed. In this model a use can occur of three forms: 1) computation use (*c-use*), when the variable is used in a computation; 2) predicative use (*p-use*), which occurs when the variable is used in a decision; and 3) communication use (*m-use*), which occurs in a inter-processes edge. A def-clear path for a variable x through the *CFG* is a sequence of nodes, n_1, n_2, \dots, n_n that do not contain a definition of x . The nodes into of *CFG* in this model also represent definitions and uses of variables.

Figure 2 shows an *ROS* graph for the example presented in the Section III. Each node represents a process and each edge represents a message between two processes. Figure 3 illustrates an *PSDU* graph, which is a part of the code of the Listings 1 and 2 with three processes exchanging messages: p_1, p_2 e p_3 . Each *CFG* represents a *callback* method and inter-processes communication edges link each graph. For example, in nodes n_2, n_4 and n_5 of the process p_1 data are defined and node n_6 they are published. When this occurs the *publish* method puts data in an output queue of the ROS and return. From this point the middleware allocates threads that delivery this message to the subscribes. For this example, the methods m_1 of process p_2 and the m_1 of process p_3 subscribe messages from m_1 of the process p_1 . The method m_1 of process p_1 subscribes messages from m_1 of p_3 .

Based on this model, the criteria for the publish/subscribe applications are:

- all-nodes-communication criterion: each process of the *PSDU* graph will be exercised at least once. The input and output interfaces of each process should be exercised by the test case set.
- all-nodes-publish criterion: all processes of the *PSDU* graph that have published messages will be exercised at least once. The interfaces provided will be exercised at least once by the test case set.
- all-nodes-subscribe criterion: all processes of the *PSDU* graph that have subscribed messages will be exercised at least once. The interfaces for the topics

that have received data will be exercised at least once by the test case set.

- all-pairs-publish/subscribe criterion: each pair of the *PSDU* graph that consists of a process that publishes data in a topic and another process that has subscribed messages from the same topic must be exercised at least once by the test case set.
- all-multiples-publish/subscribe criterion: more than two processes of the *PSDU* graph must be exercised at least once by the test case set. This criterion exercises the composition of processes. Initially, three processes are tested; next, the number of processes is increased until the maximum number of processes of the graph has been reached.
- all-m-uses criterion: each *m-use* association of the *PSDU* graph will be exercised from the last definition of the variable in a node n_i until the first use of the variable in the subscribe process ps_j , i.e., for each node n_i and each $x \in def(n_i)$, the test set must exercises a path that covers an inter-processes association w.r.t. x and $n_i \in pp_k$.
- all-sequences criterion: different input sequences are exercised for each subscribe process of the *PSDU* graph from different origins (topics). The criterion exercises the order in which asynchronous events are received.

V. EXAMPLE OF AN APPLICATION

This example is a simplified application of iRobot Roomba presented in Section III. For the application of the testing criteria, the first step is the generation of the elements required for each criterion (Table I) based on the *PSDU* graph. Table I shows some required elements. The second step is the generation of the test cases for covering the required elements. A test input for this application is data from the *Proximity Sensor* process and the expected output is a command from the *Motor Driver* process for the actuators of the robot (wheels). One test set is able to cover the required elements was generated based on the testing criteria.

Defects were inserted into the programs for the evaluation of the testing criteria in revealing faults. These defects focus mainly on the communication among the application processes. Three types of defects were inserted: 1) changes in

TABLE I. SOME REQUIRED ELEMENTS FOR THE EXAMPLE IROBOT.

Criteria	Required Elements
All-nodes-communication	$(n_6^{1,1}), (n_{11}^{2,1}), (n_8^{3,1}), (n_1^{1,1}), (n_1^{2,1}), (n_1^{3,1}), \dots$
All-nodes-publish	$(n_6^{1,1}), (n_{11}^{2,1}), (n_8^{3,1}), \dots$
All-nodes-subscribe	$(n_1^{1,1}), (n_1^{2,1}), (n_1^{3,1}), \dots$
All-pairs-publish/subscribe	$((n_6^{1,1}), (n_{11}^{2,1})), ((n_{11}^{2,1}), (n_1^{1,1})), ((n_6^{1,1}), (n_1^{3,1})), ((n_8^{3,1}), (n_1^{1,2})), \dots$
All-multiples-publish/subscribe	$((n_{11}^{2,1}), (n_1^{1,1})), ((n_6^{1,1}), (n_1^{1,1})), ((n_6^{1,1}), (n_1^{2,1})), ((n_{11}^{2,1}), (n_1^{4,1})), \dots$
All-m-uses	$(n_2^{1,1}, n_6^{1,1}, n_1^{3,1}, c_motors), (n_1^{3,1}, n_8^{3,1}, n_1^{1,2}, odometry), \dots$
All-sequences	$(p_1, t_1, t_2), (p_1, t_2, t_1), (p_4, t_2, t_1), \dots$

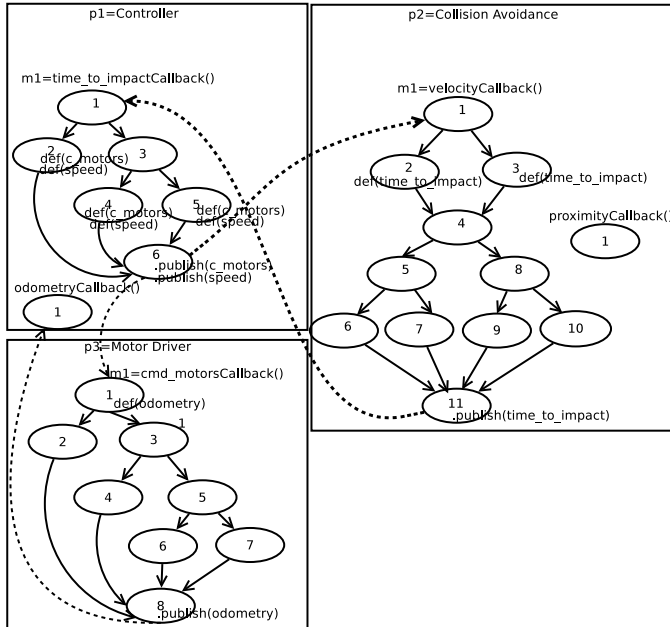


Fig. 3. Example using PSDU.

the frequency of publication of the processes, which occurs when a process sends data faster than the capacity of the receptor to take them, or otherwise; 2) non-publication of expected data by the processes, when an expected datum is not sent from a process; and 3) changes in the queue size of a topic that received data from different process, occurs when the queue size is smaller than necessary regarding the frequency of publication in a topic from different processes. Data overwritten in the queues of the topics might result in problems for the robot.

Three programs were generated from each defect, therefore, 9 programs were employed for the evaluation of the testing criteria. Table III shows the results of the execution of test cases in the programs. The first column refers to the identifiers of the programs with defects; the second column shows the results of the application for the test set. *Y* indicates an expected output, therefore, the test case did not reveal the defect and *N* indicates an unexpected output, meaning the test case could reveal the defect in the program.

The testing criteria were manually applied and no drivers or stubs were used. According to Table III, only a defect was not revealed by the test cases (Program 2c), because the inserted defect did not change the expected output. All other defects were revealed by test cases. The test cases achieved

100% coverage for the elements required. The advantage of the testing criteria proposed is they support the tester in the control of the integration testing activity. The tester choose the better strategy based on this purpose, such as to test only publish process can be used the *all-nodes-publish* criterion or if he need test integration of various processes he can use the *All-multiples-publish/subscribe* criterion. The focus of the criteria is not on the generation of the test cases, but on the support of the systematization of the activity, when the testing criteria can help in the selection of the parts of the code or specific combinations of processes to be tested during the integration testing.

TABLE II. TEST CASES.

Input	Expected outputs
-1	stop
1	stop
2	reduce
3	working
100	working

The results from manual application of the criteria shown the testing activity of various processes of a mobile robotic system can be supported by the use of integration testing criteria. The program was executed more than once to cover all required elements of *all-sequences* criterion for example. A testing tool which instruments the code, generates and integrates the CFGs for all methods can help in this task. Other specific features of *publish/subscribe* schema need more attention as race conditions, deadlocks and concurrent threads in the processing of *callbacks* will be explored in next studies with support of a testing tool.

VI. CONCLUSIONS AND FUTURE WORK

This manuscript has addressed aspects of testing mobile robotic systems and emphasized the publish/subscribe interaction schema. Specific characteristics of the publish/subscribe systems, such as non-determinism, restrictions of time and synchronization of the process have not been totally covered by the existing testing approaches.

TABLE III. RESULTS OF THE EXECUTIONS ON THE PROGRAMS WITH DEFECTS

Identifier	Test set				
	p = -1	p = 1	p = 2	p = 3	p = 100
1a	Y	Y	N	N	N
1b	Y	Y	Y	Y	N
1c	Y	Y	Y	N	N
2a	Y	Y	N	N	N
2b	Y	Y	N	N	N
2c	Y	Y	Y	Y	Y
3a	Y	Y	N	N	N
3b	Y	Y	N	N	N
3c	Y	Y	N	N	N

We have proposed a family of integration testing criteria to publish/subscribe systems. Seven testing criteria were defined for a systematic exploration of communication among components in a mobile robot. The application is represented by an *PSDU* graph generated from the ROS meta-operating system and *CFGs* for the identification of the elements required for these criteria. The defects identified would not usually be revealed with the use of simulations only or unit testing because the focus of our model is on exploring the internal characteristics of each process and its influences in the publish/subscribe schema. An example shown the ability of our approach to support the integration testing activity.

We intend to use more complex systems as case studies to test concurrent aspects when more intense computation is involved. In addition comparing our testing criteria with other testing approaches (simulation using VSTs, for instance). We are currently developing an coverage analysis tool to support the integration testing criteria.

ACKNOWLEDGMENT

The authors acknowledge the Brazilian funding agencies FAPESP, under processes 2013/03459-4 and 2013/01818-7 and CAPES, under process DS-8435201/M, for the financial support provided for this research.

REFERENCES

- [1] Ros/concepts. <http://wiki.ros.org/ROS/Concepts>. [Accessed 18/10/2014].
- [2] A. G. Araújo. ROSint - integration of a mobile robot in ROS architecture. Master's thesis, University of Coimbra, Coimbra, Portugal, 2012.
- [3] Y. Ben-Asher, Y. Eytani, E. Farchi, and S. Ur. Producing scheduling that causes concurrent programs to fail. In *Workshop on Parallel and Distributed Systems: Testing and Debugging*, pages 37–39, 2006.
- [4] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback. Orca: A component model and repository. In *Software Engineering for Experimental Robotics*, volume 30, pages 231–251, 2007.
- [5] H. Bruyninckx. Open robot control software: the OROCOS project. In *Int. Conference on Robotics and Automation*, volume 3, pages 2523–2528, 2001.
- [6] Q. Chen, L. Wang, Z. Yang, and S. Stoller. HAVE: Detecting atomicity violations via integrated dynamic and static analysis. In *Fundamental Approaches to Software Engineering*, volume 5503, pages 425–439, 2009.
- [7] Z. Chen, X. Li, J. Y. Chen, H. Zhong, and F. Qin. SyncChecker: Detecting synchronization errors between MPI applications and libraries. In *Parallel Distributed Processing Symposium (IPDPS)*, pages 342–353, May 2012.
- [8] P. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.
- [9] L. C. Fernandes, J. R. Souza, G. Pessim, P. Y. Shinzato, D. O. Sales, V. Grassi Jr., K. R. L. J. Branco, F. S. Osorio, and D. F. Wolf. CARINA intelligent robotic car: Architectural design and implementations. *Journal of Systems Architecture*, 2013.
- [10] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. J. Mataric. Most valuable player: a robot device server for distributed control. In *Int. Conference on Intelligent Robots and Systems*, volume 3, pages 1226–1231 vol.3, 2001.
- [11] M. Y. Jung, A. Deguet, and P. Kazanzides. A component-based architecture for flexible integration of robotic systems. In *Int. Conf. on Intelligent Robots and Systems*, pages 6107–6112, Oct 2010.
- [12] J. S. Kang and H. S. Park. RPIST: Required and provided interface specification-based test case generation and execution methodology for robot software component. In *Int. Conference on Ubiquitous Robots and Ambient Intelligence*, pages 647–651, 2011.
- [13] S. S. Kang, S. W. Maeng, S. W. Kim, and H. S. Park. SITAT: Simulation-based interface testing automation tool for robot software component. In *Int. Conference on Control Automation and Systems*, pages 1781–1784, Oct 2010.
- [14] H. K. N. Leung and L. White. A study of integration testing and software regression at the integration level. In *Conference on Software Maintenance*, pages 290–301, 1990.
- [15] J. H. Lim, S. H. Song, T. Y. Kuc, H. S. Park, and H. S. Kim. A hierarchical test model and automated test framework for RTC. In *Int. Conf. on Future Generation Information Technology*, pages 198–207, 2009.
- [16] C. Lucas, S. Elbaum, and D. S. Rosenblum. Detecting problematic message sequences and frequencies in distributed systems. In *Int. Conf. on Object Oriented Programming Systems Languages and Applications*, pages 915–926, New York, NY, USA, 2012.
- [17] R. S. Pressman. *Software Engineering: Practitioner's Approach*. McGraw-Hill, 6 edition, 2005.
- [18] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [19] P. S. L. Souza, S. R. S. Souza, and E. Zaluska. Structural testing for message-passing concurrent programs: an extended test model. *Concurrency and Computation: Practice and Experience*, 26(1):21–50, 2014.
- [20] S. R. S. Souza, P. S. L. Souza, M. C. C. Machado, M. S. Camillo, A. S. Simão, and E. Zaluska. Using coverage and reachability testing to improve concurrent program testing quality. In *Int. Conf. on Software Engineering and Knowledge Engineering*, pages 207–212, Eden Roc Renaissance Miami Beach, USA, Jul 2011.
- [21] S. R. S. Souza, S. R. Vergilio, P. S. L. Souza, A. S. Simão, and A. C. Hausen. Structural testing criteria for message-passing parallel programs. *Concurrency and Computation: Practice and Experience*, 20(16):1893–1916, 2008.
- [22] A. S. Tanenbaum. *Modern operating systems*. Second edition, 2001.

Embedded Real Time Blink Detection System for Driver Fatigue Monitoring

Soheil Salehian* & Behrouz Far†
Department of Electrical and Computer Engineering,
University of Calgary,
Calgary, Canada.
Email: *ssalehia@ucalgary.ca, †far@ucalgary.ca

Abstract—Fatigue induced vehicle accidents have seen an increase in the last few decades. Fatigue monitoring using non-invasive and real time image processing and computer vision techniques have shown great promise and are an active research area. To that extent, in the proposed work a blink detection algorithm is proposed that serves as a visual cue that may be correlated to the state of fatigue of the driver. Using a complimentary but independent approach, shape analysis and histogram analysis are carried out in parallel to perform the blink detection task. Close to real time performance and a high level of accuracy in controlled settings show great promise of such approach in enhancing the monitoring of the driver's blinking patterns. One of the main constraints of using such algorithm in a real world setting is the minimized processing time required to allow for sufficient driver response time. In this work implementation of the algorithm is described using optimization techniques to meet such latency requirements. The validation of the algorithm was carried out by visual inspection of the video sequences in terms of precision and accuracy. The presented blink detection algorithm has a precision rate of 84% and an accuracy rate of 69% obtained through using 12 sequences of different duration videos in varying lighting conditions using a small sample of participants.

Index Terms—computer vision, blink detection, driver fatigue, image processing.

I. INTRODUCTION

The alarming number of traffic accidents due to driver fatigue accounts for more than half of all truck collisions in the United States [1]. Diminished levels of attention caused by fatigue, increases response time while in more severe cases it may result in short lapses of sleep by the driver. Research has shown that after 2-3 hours of constant driving, fatigue plays an important factor in slowing decision making and perception of the driver of the vehicle. More recently, a study by the National Sleep Foundation in the US showed that in their study, more than 51% of adult drivers with drowsy symptoms had driven a vehicle and 17% had momentarily fallen asleep while driving [2]. It is estimated that 1,200 deaths and 76,000 injuries are due to fatigue induced accidents annually. Due to the recent attention to fatigue related crashes, fatigue detection systems have become an active area of research.

There has been substantial work on characterizing driver fatigue based on various models. Dinges et al. [3] showed that physiological signals such as the electroencephalography (EEG) and the electro-cardiogram (ECG) can be used to

measure fatigue. Other less intrusive methods using physical information of the vehicle combined with the patterns of driving has also been researched extensively with limited success[4]. Although the most accurate results have been reported using physiological instrumentation, such systems are not practical as they require initial setup which is a hassle for the driver [5], [6]. With non intrusiveness constraints, another category of methods that has become an active research area is non-intrusive online monitoring of the driver using computer vision. In this category, "visual cues" such as gaze, head movement, and eye blink rate is tracked in order to accurately estimate the state of the driver. These computer vision techniques aim to extract visual fatigue related characteristics in real time using image/video processing. For instance, Boverie et al. [7] developed a system to correlate eyelid movement to estimate the degree of vigilance of the driver. While others such as Ueno et al. [4] looked at methods to measure the degree of openness of the eyelids to make the fatigue characterization. The majority of such early studies involved strictly controlled environments, lighting conditions and line of sight for the extraction process to work properly. Recent clinical research on the effectiveness of blinks as strong indicators of fatigue [8], make blink detection a strong candidate that is the basis of the following work.

Although most of previous work has focused on the ability of the vision system to correctly detect blinks in various lighting conditions, in real world software vision systems there are response time requirements that dictate the effectiveness of such systems. Studies by Muttart [9] have quantitatively examined the driver response times based on various real conditions on the road. Their conclusion suggested that there is a variety in response time in drivers that is obviously correlated to driver speed and conditions. Therefore, it is of note to mention that the response time of a developed computer vision system should be minimized as much as possible to account for this variation in driver population and allow for sufficient time of reaction in real settings. From the moment the system has the image data this time is labelled *processing time* and is required to respond in less than an estimated 900ms. It is the focus of the following work to not only develop the algorithm but enhance in its performance on an implementation in an embedded system with real time constraints.

In the presented chapter, an eye blink detection algorithm is proposed using machine learning and image processing techniques in an effort to enhance the robustness of blink detection as an important part of a driver fatigue monitoring system. The contribution of this work includes two complementary algorithms that exploit different information in each image/frame in order to arrive at a more robust estimation of the driver blink rate along with their concurrent implementation on an embedded system achieving real time requirements. The remainder of the paper is organized as follows: Section II provides a detailed explanation of the proposed algorithm methodology. The implementation details and optimization required to enhance performance of the vision system is described in Section III. The results of the critical steps of the algorithm and the validation methodology is presented in Section IV. A discussion of the results of the algorithm in a video processing setting is part of Section V and finally, conclusions on effectiveness and limitations of our approach along with future work is outlined in section VI.

II. METHODS

In the following section, the eye blink algorithm is described with a detailed discussion on key sections of the algorithm such as: face detection, eye detection, pre-processing of the region of interest (ROI), shape analysis, complimentary histogram analysis method and combination of their outputs. The algorithm was designed using a set of real-time video captures in various lighting conditions for robustness verification.

A. Algorithm Overview

The high level flow of the proposed algorithm, initiates with detecting the face area using Haar-features. These features are extracted using a Haar classifier that has been trained with frontal face images. Once the face area is located, a second classifier using similar Haar-features finds the eye band area for both eyes. This eye band area is the ROI that will be processed by two separate eye blink detection methods. In one method, called pipeline A, the gray scale image of the initial frame is used as input. Then, the edges within the ROI are detected using the canny edge detector with a 3×3 Gaussian blur filter to remove noise. This edge detection works well because there is a strong contrast between the iris and the choroid.

The algorithm proceeds by doing contour analysis, where the various large contours of the ROI are examined further. Due to the elliptical shape of the eyes while being open, an ellipse fitting is performed to identify the eyes as open. Upon closure of the eyes, the number of ellipses, corresponding to each eye, in the image reduces greatly which indicates that a blink may have happened.

However our experimentation shows that contour information may not be sufficient in robustly detecting the blinks. Therefore we have chosen to implement a second computationally efficient method that concurrently enhances the detection outcome.

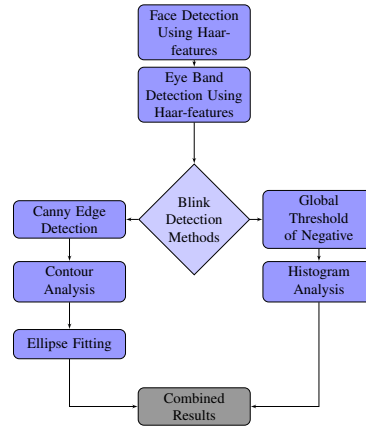


Fig. 1: Flow chart representation of the proposed blink detection algorithm. Please note the two parallel sub-algorithms and the combination of their results.

In parallel, a secondary method (pipeline B) looks at non spatial information of the ROI. This second method, takes the negative of the frame and uses a simple threshold to globally threshold high pixel values (which will include the surrounding areas around the eyes and the eye structure). The histogram of such binary image has a bimodal shape with two impulses. It can be observed that blinking reduces the number of white pixels because momentarily the eyelids will cover the eyes and there is a shift of pixels from the high end of the histogram to the low end which is detected by the algorithm. The fusion of the results of contour/shape analysis with histogram analysis allows for the detection of eye blinks.

A flowchart representation of the algorithm overview is presented in Fig. 1. The various stages of the algorithm is explained in more detail in the following sections.

B. Face Detection

The following section presents the learning-based method used in detecting the face area in our blink detection algorithm. Learning based methods use training samples in combination with statistical and machine learning models that have been shown to be effective in detecting facial features [10]. One main advantage of learning methods is their ability to adapt to various scenarios given adequate and large training sets. Variety of lighting conditions, driver demographics and other features specific to the driving of the vehicle can be included in the training set in order to increase accuracy and robustness of the face detection process.

Viola et al. [11] proposed a set of features named Haar-like features due to similarity to Haar wavelet basis functions. Their algorithm has gained popularity due its robust and computationally efficient property for object detection specifically in the face detection domain. Haar-like features use the change in contrast of adjacent rectangular groups of pixels instead of the pixel's own intensity values. The variance between the neighbourhoods surrounding the pixel are used to identify areas of high and low intensity values. Different number of

grouping of such basis functions based on their variance can result in detecting different types of features such as edge, line or center-surround features [12].

The simplicity of these features allow for scaling and therefore scale-invariant detection of face region in the frame. Viola et al. [11] showed that for a rather small image, the total number of such elementary features is in the order 180,000 which may be impractical to calculate [12]. However, for accurate object detection, they noted that not all features are required. By transforming the image into what they called an "integral image", any of the haar features is able to be computed at any scale in constant time. The construction of the features is initiated by generating the integral image. The integral image intermediate representation (iI) of original image (I) at x,y contains the sum of pixels above and to the left at x',y' which can be formally defined as:

$$iI(x,y) = \sum_{x' \leq x, y' \leq y} I(x',y')$$

Then the cumulative row sum $s(x,y)$ is:

$$s(x,y) = s(x,y-1) + i(x,y)$$

Then the integral image can be re-written in terms of the cumulative row sum as:

$$iI(x,y) = iI(x-1,y) + s(x,y)$$

Using this simple technique in generating the integral image, all the combination of rectangle feature sets may be constructed which makes feature generation computationally efficient. These features are sensitive to presence of edges, bars, and simple structures with only horizontal, vertical and diagonal orientation [11]. Specifically in the case of face detection, it was noted that some features are more effective than others based on exploiting the property of the region of the eyes that is often darker than the region of the nose and cheek (with a higher variance in the eye region) and similarly darker region of the eyes from the bridge of the nose.

The effect of different lighting conditions are important in a variance based method and therefore during the training it was addressed by a variance normalization procedure defined as:

$$\sigma_w^2 = \mu_w^2 - \frac{1}{N} \sum p_w^2$$

Where the variance σ^2 of window w is defined in terms of mean of the window (μ_w) and the sum of squared pixels p of window w . The summation is calculated using the integral image procedure previously described. It is important to note that such normalization in the training procedure is inherently needed in the detection phase as well.

With the feature generation phase complete, a learning method is required in order to perform a classification function. The AdaBoost learning algorithm is used for both tasks of feature selection and the training of the classifier [13]. Using



(a) Open eyes ROIs

(b) Closed eyes ROIs

Fig. 2: Results of face (blue) and eye band (green) ROI detection using Haar-classifiers. The classifier has worked well in identification of the eye ROI regions while the eyes are closed in (b).

the Adaboost weak learner procedure, each classifier can only depend on a single feature and cascading of such classifiers allow for a robust method for scale invariant object detection.

A large reduction in the number of non-contributing features, and its excellent generalization performance allows AdaBoost to be used in a cascaded format that forms the cascade classification of haar-features. The cascading of classifiers allows for training each classifier using AdaBoost and adjusting each classifier's threshold and weights to minimize false negatives using error minimization.

The results have proven to have a high accuracy rating and a subsequent better performance as the cascading continues. This makes the haar-like cascaded classifier method ideal for the face detection task of our real time blink detection algorithm.

C. Eye Band Detection

The eye band detection method to identify the eyes in each frame uses the same methodology as the face detection mechanism via Haar-like features and AdaBoost combination. Beyond the technical aspects of the classifier mechanism, for the eye detection few considerations are worth mentioning:

- By finding the face region in each frame, the ROI for eye detection becomes smaller and the performance of the classifier increases dramatically.
- A separate training set for eyes is used to train the classifier. In the case of this work in its current form, a pre-trained classifier was used which performed adequately for the eye detection task.
- The decision was made to detect both eyes as an "eye band" as the trained classifier performed best when both eyes were facing the camera.

Fig. 2 demonstrates the results of simultaneous face and eye detection for frames corresponding to both open and closed eyes using the described procedure.

D. Canny Edge Detection

Edge detection is an important procedure and a first step in identifying objects of interest in the image. Once the ROI has been identified, edge detection allows for structural analysis

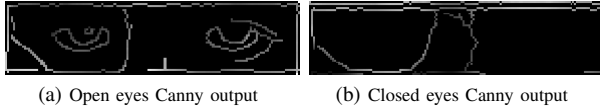


Fig. 3: Results of canny edge detection using $lowThreshold = 60$ and $maxThreshold = 140$ for both open and closed cases. Please note that while closed, the detector has only detected the shadow area around the eyes.

inside the ROI which in the case of this work is the eye band detected in the previous stage. There are a number of edge detectors that may be used depending on the desired structural properties. In the proposed algorithm, the popular Canny edge detector algorithm [14] was selected due to its following characteristics:

- 1) Robust detection: in the blink detection application, the probability of detecting real edges need to be quite high despite high noise levels in each frame.
- 2) Computationally inexpensive: due to the real time nature of the application, high performance was a secondary but important deciding factor.
- 3) Step edges: the strong variance between the eye region and skin (both horizontally and vertically) can be characterized as step edges which the canny algorithm was originally designed for [14].

The resulting output of this stage, is a binary image that identifies all the corresponding edges in the eye band area. Fig. 3 shows the effect of the operator in open and closure sample cases described previously. It is worth mentioning that the low threshold for the method was proven to be critical in detecting important edge structure between the eye choroid and the pupil. A severely low threshold would mark many details of the eye band as edges which was sub-optimal for the blink detection algorithm. The thresholds were chosen empirically based on the training set frames during development.

E. Contour Extraction & Analysis

By finding the strong edges using the previous operation in the eye band region, the structural information of the ROI is ready for further analysis in detecting the eye regions. The goal of this proposed stage is to find an approximation of all the contours that are present in each frame. The following section explains in detail how contour extraction from the edge information is accomplished and the corresponding analysis that is carried out on each contour.

The contour extraction operation used in our algorithm is based on the work of Suzuki et al. [15] where a topological analysis is done on the contours found by what is known as "border tracing" based on earlier work by Rosenfeld et al.[16] and the utilization of Freeman's chain codes [17].

Once all contours are traced using the above algorithm, the operation proposed by Suzuki et al. derives a sequence of coordinates on each contour and constructs a topological ordering of such coordinates. It was shown that using such

technique, both outer contours and inner contours (holes) can be effectively labeled and the topological analysis can lend to accurate categorization and discrimination of enclosing contours vs. inner contours. In the case of our blink detection algorithm, due to the large size of both eye regions in the band area, it was desired to analyze the outer contours in each frame. The proposed algorithm exploits the fact that during the blinking motion, larger contours of the eye will be deformed and disappear rapidly and hence extracting the contours is a first step in monitoring the blinking. Once the contours are extracted, the proposed algorithm proceeds by calculating the area of each contour for further analysis. It was observed that discriminating the large contours (such as large reflections due to sever lighting conditions) or smaller extracted contours (due to poor edge extraction) based on area threshold was an effective method in keeping only contours related to the eye region. The area threshold is adaptive and based on the size and resolution of the eye band frame.

F. Ellipse Fitting

Once the corresponding contours to the eye region (the choroid and iris sections) have been identified, shape analysis is the next stage of the proposed algorithm. In this section, the ellipse fitting procedure and some of the assumptions and criteria of the analysis is discussed in more detail. The intuition behind the procedure is based on a *priori* that the eye region contours have an elliptical shape. Fitzbibbon et al. [18] evaluated various methods of fitting data to conic sections based on assumptions of isotropic normally distributed noise and incomplete contours. Their work is of particular interest in our application due to its analysis of performance in terms of algorithm complexity and computation in the task of ellipse fitting using least-squares as a distance metric. Fitzbibbon et al. showed that based on their experimentation evaluation of the popular conic fitting algorithms with strong variants of noise, orientation, and occlusion, that least-squares based algorithm of statistical distance (also known as BIAS [19]) has a good tradeoff between performance and accuracy in fitting contour points to an ellipse even with the presence of outliers due to high noise and discontinuities.

During the shape analysis stage of the blink detection algorithm (implementation in OpenCV), all fitted ellipses had to meet a discrimination criteria to be included for further stages. The orientation of the fitted ellipses become important in distinguishing eyes from other ellipses. It can be observed that contours of the eye region has an orientation close to the horizontal line with some varying angle, θ , assuming that the driver is not orienting their head substantially. Choosing an appropriate threshold on θ allows the algorithm to only include ellipses resembling the contours of the eye region (θ was empirically found to be 10°). Another discriminant which was used to avoid fitting all possible contours was the contour area. Fig. 4 shows the results of ellipse fitting with and without constraints in both open and eyes closed cases. Please note that the fitted small ellipse in Fig. 4 (b) has been discarded using a contour area threshold based on the resolution of the frame

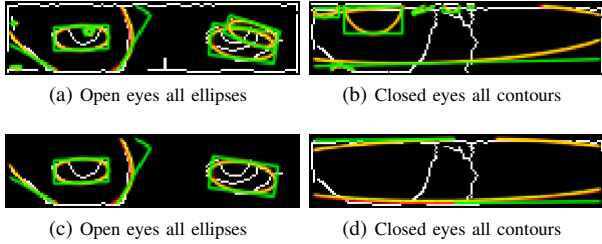


Fig. 4: Results of the ellipse fitting procedure using discriminant of $\theta = 10^\circ$ and $contourArea = 20$ pixels. The red and yellow line are the ellipse fitting procedure while the green box is the rectangular fit for orientation analysis.

in Fig. 4 (d). The effect of orientation discriminant can be observed in between Fig. 4 (a) and (c).

With the shape analysis in place, this pipeline of the algorithm (pipeline A) is able to classify the eyes in the frame as open or closed. During the blinking motion, the monitored number of allowable ellipses at each frame has a reduction of 2 or more which will indicate that the eyes have closed and therefore the frame can be marked accordingly.

Although this pipeline works in most test cases, it was observed that due to variations of sampling from different experimented cameras and the variability in lighting conditions, the pipeline requires a complementary synchronization mechanism to overcome some of these shortcomings. A second complimentary pipeline (pipeline B) was developed to run in parallel which will be discussed in the following section.

G. Global Thresholding Of Negative

To independently complement the shape/contour analysis pipeline discussed in the previous sections, pipeline B was developed using a rather simple thresholding technique. It was observed that during the blinking motion, when the eyelids cover the choroid and iris there is a change in the number of pixels that represent the skin. By exploiting this idea, a global threshold on the negative of the gray scale frame is used to detect the eye region and its approximate surroundings. The global thresholding is formally defined as [17]:

$$g(m, n) = \begin{cases} 0 & \text{if } f(m, n) \leq \tau \\ 255 & \text{if } f(m, n) > \tau \end{cases}$$

where the resulting binary image, $g(m, n)$, is based on the global thresholding operation on the original gray scale image $f(m, n)$ with threshold τ . This procedure renders robust results with prior knowledge of the lighting conditions and range of skin pixel values.

H. Histogram Analysis

Using the resulting binary image, simple histogram analysis is used to monitor two groups of pixels in the image. One group belongs to the background and the other group is the approximate eye region. During the blinking motion, there is a substantial change in the number of high intensity pixels and a

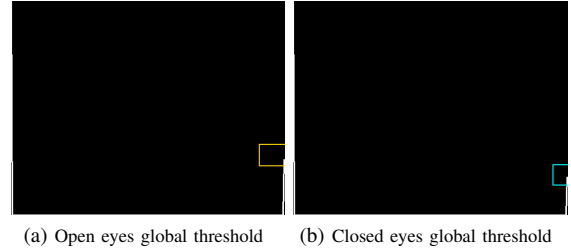


Fig. 5: Results of global thresholding, $\tau = 185$ on the negative of frames for both open and closed cases. Please note the decrease of the number high intensity pixels in (b).

comparison of this number with the previous frame has shown to be robust in the preliminary results of this work. Using this difference, $\%d$, the frame is classified as closed only if the difference is greater and equal to 20% of the previous frame's number of high intensity pixels (found empirically). Fig. 5 shows the sudden drop of the number of high intensity pixels during the blinking motion.

I. Merging of Results

In this final stage of the algorithm, the results of the shape analysis/elliptical fitting (pipeline A) and the results of global thresholding (pipeline B) is merged for each frame. If results from both pipelines match to be closure, the frame is classified as a detected blink. If there is a discrepancy among the pipelines, the result is marked as an open eye.

III. REAL-TIME IMPLEMENTATION

In this section some of the details of the implementation of the proposed algorithm is presented. As previously mentioned, strict real time requirements are present in order for the system to fulfill the minimization of *processing time* of 900 ms and allow for slowest human driver response time in the process. The embedded system chosen for this work was the Nvidia Jetson TK1 which includes an ARM Cortex A-15 dual core processor and a Nvidia GPU for embedded vision applications on the Ubuntu 14.04 L4T platform. Although our development environment was similar on Linux 14.04 on x64 machine, during the porting process on the target embedded system, there were optimization to be considered. With the latency constraint of *processing time* being 900 ms it was important to optimize CPU code.

One of the main steps in the algorithm that is used by both pipeline A and pipeline B extensively was the color to gray scale conversion. With the target platform being an ARM based CPU with ARM's NEON [20] capabilities which allows for single instruction multiple data (SIMD) operations, the approach chosen was to enhance gray scale conversions using NEON intrinsic instructions.

The optimized implementation of gray scale conversion using NEON intrinsics is shown below:

```
void neon_rbg_gray (uint8_t * __restrict dest,
uint8_t * __restrict src, int numPixels)
{
```

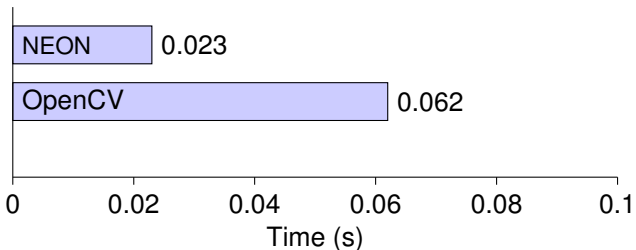


Fig. 6: Comparisons of our Neon optimized RGB to gray scale conversion compared to native OpenCV C++ performance.

```

int i;
// 8x8 Neon registers are filled
// Red channel multiplier
uint8x8_t rfac = vdup_n_u8 (77);
// Blue channel multiplier
uint8x8_t gfac = vdup_n_u8 (151);
// Green channel multiplier
uint8x8_t bfac = vdup_n_u8 (28);
int n = numPixels / 8;

// Conversion in 8 pixel chunks
for (i=0; i < n; ++i)
{
    uint16x8_t temp;
    uint8x8x4_t rgb = vld4_u8 (src);
    uint8x8_t result;

    temp = vmull_u8 (rgb.val[0],    bfac);
    temp = vmlal_u8 (temp, rgb.val[1], gfac);
    temp = vmlal_u8 (temp, rgb.val[2], rfac);

    result = vshrn_n_u16 (temp, 8);
    vst1_u8 (dest, result);
    src += 8*4;
    dest += 8;
}
}

```

Fig 6 shows some of the earliest results with comparison to OpenCV’s native C++ implementation. As it can be observed a reduced factor of 2.5x allows for the system to meet the latency requirements of *processing time* of 900ms.

IV. RESULTS

The following section outlines the results obtained through experiments of this work. The proposed algorithm was developed and visually validated using live video processing of a 720p webcam at 20 frames per second using the OpenCV C++ APIs. For validation of this work, our preliminary validation procedure was carried out from a set of sequence of frames picked from various recordings in different lighting conditions and a small sample of subjects. The sampling included a relatively even distribution of the three possible cases of: open eyes, closure motion, and closed eyes. As it is evident, the algorithm initially needs to detect the face and eyeband ROI region accurately and reliably prior to moving on to the later stages for blink detection. By constraining the distance away from the camera and the head orientation of the driver, it was found that the Haar-feature classifier performs well based on our qualitative analysis of frames through out development and on the sequence frames used for validation.

The distribution of the sequences which can be observed in Table I, are as follows: sequences $s1 - s5$ were based

TABLE I: Results of proposed blink detection algorithm in different sequences of frames in moderate ($s1 - s5$), high ($s6 - s9$) and low illumination ($s11 - s12$) conditions. Accuracy and precision of each sequence and the total average has been shown.

Sequence #	Frames	AP	TP	FP	FN	Accuracy	Precision
s1	242	16	15	1	0	0.9375	0.9375
s2	200	12	12	0	0	1.0000	1.0000
s3	193	8	8	0	1	0.8889	1.0000
s4	300	4	3	1	2	0.5000	0.7500
s5	275	21	19	2	1	0.8634	0.9048
s6	250	14	12	2	9	0.5218	0.8571
s7	220	9	8	1	6	0.5333	0.8889
s8	244	11	9	2	8	0.4737	0.8182
s9	207	7	5	1	1	0.7142	0.7143
s10	190	10	6	4	2	0.5000	0.6000
s11	202	12	9	3	2	0.6428	0.7500
s12	248	14	13	1	3	0.7647	0.9286
Average						0.6942	0.8458

on moderate illumination conditions, $s6 - s9$ were realized in highly illuminated and $s10 - s12$ were in low illuminated conditions. The results including the accuracy and precision of each sequence have been based on the assumption that the eyeband ROI detection procedure has been successful (as the algorithm will not carry on if the eye band ROI is not detected). True positives (TP) include all cases that both eyes were closed and the system was able to detect the blink. The cases of detected false blinks when they were either open or the eye band ROI was not even detected, is labeled as false positives (FP). The percentage of inability of the algorithm to detect closure while detecting the eyeband ROI region is included under the false negatives category (FN). The number of actual positives (AP) was measured by repeated review of each sequence via only visual inspection at this time.

A noteworthy detail of the implementation of the algorithm is that if for any reason the eye ROI is not found, the algorithm stays idle without exiting the program. The detection resumes as normal once the ROI is found, which helps in eliminating non relative frames in the validation procedure. However as it was discussed previously, in almost all cases 100% of frames were included due to both the highly robust performance of the face/eye detection stages and also the controls of the experiments.

V. DISCUSSION

The preliminary results of the validation of the algorithm, show promise of the proposed complimentary approach of using shape analysis in parallel with histogram analysis. This is apparent in the accuracy and precision results in sequences $s1 - s5$ of Table I, where the small sample of participants in moderate and consistent lighting conditions. However, it was observed that both the canny edge detector and global thresholding methods are sensitive to both highly illuminated environments (observed in sequences $s6 - s8$) and low illumination conditions (sequence $s10 - s11$).

Specifically, the accuracy of sequences s_6 – s_8 has decreased substantially in comparison to the moderate lighting conditions of the first set of sequences. The analysis shows that this might be due to the performance of the canny edge detector. It was observed that the low threshold and maximum threshold during its hysteresis phase, needed to be adjusted manually in order to improve the performance of the edge detection in low illumination conditions. To improve on this shortcoming, histogram equalization of the original gray scale was applied which has helped in reducing low illumination effects. The preliminary results of inclusion of the histogram equalization operator is shown in sequence s_{12} for reference with improvements in accuracy with a slight negative effect on precision. Further validation is required to concretely conclude the effect of the operator.

For high illumination cases, the histogram threshold had to be re-adjusted as it was observed that due to high illumination around the eyes, the thresholding results would include a larger area in the proximity of the eyes and hence a higher number of high illuminated pixels. This higher number effects the histogram analysis which means that the difference between the open eye frame and a closed eye frame may be smaller than the predefined difference $\%d$ set in average illumination cases. The results show degradation of performance in sequences s_{10} – s_{11} . To improve on the robustness of choosing $\%d$, a normalization factor may be used. The ratio of the number of high pixels to low pixels seem to have eliminated this issue in controlled conditions resulting in performance improvements especially to precision as it can be seen in the results for s_9 . However extensive validation is required to analyze the full effect of this correction.

In terms of computational performance, our implementation allows to meet the minimum driver response time requirements of the real time system the algorithm is designed for. More extensive profiling shows other areas of improvements such as using the GPU to enhance the performance of edge detection using the Canny edge detector.

VI. CONCLUSION & FUTURE WORK

This work presented a blink detection algorithm based on two complimentary, but independent approaches using shape and histogram analysis. The monitoring of the driver's blink patterns was performed in near real time using efficient image and computer vision techniques. The preliminary results in terms of total accuracy and precision rates indicate that the current approach can be useful in monitoring blink detection for fatigue. The algorithm requires training in more varying lighting conditions in order to be robust. Future work will include improvements to the image acquisition system such as using an infra red camera and also additional preprocessing techniques such as gamma correction or histogram equalization. Furthermore the inclusion of adaptive methods in the edge detection step and also in the global thresholding stages will be part of the continuation of this work. Using similar techniques in identifying other visual cues such as facial

expressions and yawning may enhance the accuracy of a well defined driver fatigue detection in the future.

ACKNOWLEDGMENT

This work has been supported by the Alberta Innovates Technology Futures and the University of Calgary. The authors would like to thank Dr. Rangayyan for the technical discussions during this work.

REFERENCES

- [1] Q. Ji, Z. Zhu, and P. Lan, "Real-time nonintrusive monitoring and prediction of driver fatigue," *Vehicular Technology, IEEE Transactions on*, vol. 53, no. 4, pp. 1052–1068, 2004.
- [2] W. Wierwille, "Overview of research on driver drowsiness definition and driver drowsiness detection," in *Proceedings: International Technical Conference on the Enhanced Safety of Vehicles*, vol. 1995. National Highway Traffic Safety Administration, 1995, pp. 462–468.
- [3] D. Dinges and M. Mallis, "Managing fatigue by drowsiness detection: Can technological promises be realized?" in *International Conference On Fatigue and Transportation, 3RD, 1998, Fremantle, Western Australia*, 1998.
- [4] H. Ueno, M. Kaneda, and M. Tsukino, "Development of drowsiness detection system," in *Vehicle Navigation and Information Systems Conference, 1994. Proceedings., 1994.* IEEE, 1994, pp. 15–20.
- [5] M. Kaneda, H. Iizuka, H. Ueno, M. Hiramatsu, M. Taguchi, and M. Tsukino, "Development of a drowsiness warning system," in *Proceedings: International Technical Conference on the Enhanced Safety of Vehicles*, vol. 1995. National Highway Traffic Safety Administration, 1995, pp. 469–476.
- [6] S. Saito, "Does fatigue exist in a quantitative measurement of eye movements?" *Ergonomics*, vol. 35, no. 5-6, pp. 607–615, 1992.
- [7] S. Boverie, A. Giralt, J. Lequellec, and A. Hirl, "Intelligent system for video monitoring of vehicle cockpit," SAE Technical Paper, Tech. Rep., 1998.
- [8] R. Schleicher, N. Galley, S. Briest, and L. Galley, "Blinks and saccades as indicators of fatigue in sleepiness warnings: looking tired?" *Ergonomics*, vol. 51, no. 7, pp. 982–1010, 2008.
- [9] J. W. Muttart, "Quantifying driver response times based upon research and real life data," in *3rd International Driving Symposium on Human Factors in Driver Assessment, Training, and Vehicle Design*, vol. 3, 2005, pp. 8–29.
- [10] A. A. Lenskiy and J.-S. Lee, "Drivers eye blinking detection using novel color and texture segmentation algorithms," *International Journal of Control, Automation and Systems*, vol. 10, no. 2, pp. 317–327, 2012.
- [11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–511.
- [12] P. I. Wilson and J. Fernandez, "Facial feature detection using haar classifiers," *Journal of Computing Sciences in Colleges*, vol. 21, no. 4, pp. 127–133, 2006.
- [13] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Computational learning theory*. Springer, 1995, pp. 23–37.
- [14] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.
- [15] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.
- [16] A. Rosenfeld and A. C. Kak, *Digital picture processing*. Elsevier, 1982, vol. 2.
- [17] R. M. Rangayyan, *Biomedical image analysis*. CRC press, 2004.
- [18] A. W. Fitzgibbon, R. B. Fisher *et al.*, "A buyer's guide to conic fitting," *DAI Research paper*, 1996.
- [19] K.-i. Kanatani, "Statistical bias of conic fitting and renormalization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 3, pp. 320–326, 1994.
- [20] C. Pujara, A. Modi, G. Sandeep, S. Inamdar, D. Kolavil, and V. Tholath, "H. 264 video decoder optimization on arm cortex-a8 with neon," in *India Conference (INDICON), 2009 Annual IEEE*. IEEE, 2009, pp. 1–4.

A Smartphone-based System for Automated Congestion Prediction

Lance Fiondella¹, Swapna S. Gokhale², and Nicholas Lownes³

¹Dept. of Electrical and Computer Engineering, University of Massachusetts, Dartmouth, MA 02747

²Dept. of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269

³Dept. of Civil and Environmental Engineering, University of Connecticut, Storrs, CT 06269

Abstract

Accurate collection of traffic data is essential, tactically for efficient highway operations and strategically for capacity planning purposes. Currently, this traffic data is collected by physical sensors, which suffers from many limitations. These sensors gather limited measurements of vehicle speeds and times from their fixed locations and cannot systematically acquire mobility dynamics of individual vehicles and the interactions among them. Also, because deploying and maintaining physical sensors is expensive and time consuming, typically they are installed to measure traffic only on the busier road segments within a transportation network. To overcome these issues, this paper reports on a general-purpose, location-aware, smartphone-based traffic monitoring system as a simple and an inexpensive alternative to collect dynamic vehicle data extensively through a transportation network. A privacy-preserving smartphone application was developed, deployed, and tested on the network surrounding the University of Connecticut (UConn) in Storrs. It securely transmits individual trip data over the Internet to the servers hosted at UConn. Preliminary experimentation suggests that crowdsourcing the collection of traffic data with smartphones can be cost effective and can lead to richer data sets spanning the entire web of roadways.

1 Introduction

Presently, many highway operations centers rely on physical sensors deployed at fixed locations to collect traffic data. Large volumes of video feeds collected by these sensors are then monitored and analyzed by human experts to identify evolving conditions that may disrupt traffic flow, including rush-hour congestion and accidents. These experts must also manually enter traffic alerts to be displayed on signs to caution motorists of the present and changing conditions that may negatively impact travel. Collecting traffic data using physical sensors suffers from three drawbacks:

- Sensors can collect measurements only from a set of fixed locations, with no means to either record or infer traffic conditions between these discrete collection

points. Capturing traffic data on a continuum, however, can offer better insights into what is transpiring between these discrete locations. Based on such continuous data, we can infer traffic conditions that vehicles will encounter through the network, accurately predict impending congestion before it becomes severe, and issue warnings to motorists to seek alternate routes. Such proactive warnings may be superior compared to the current, reactive alerts that cannot prevent motorists from being engulfed in heavy congestion.

- Physical sensors collect measurements on the collective, aggregate traffic behavior rather than gathering data on the mobility dynamics of individual vehicles including their acceleration, deceleration, and steering. They also cannot report on the interactions among the vehicles. Vehicle-level measurements such as the acceleration and deceleration could be correlated to the safety of a road segment, while inter-vehicle interactions could offer new insights into how congestion forms and at what rate.
- Deploying and maintaining sensors on roadways can be expensive, time consuming, and hazardous. Because of these monetary and safety constraints, physical sensors may be installed only on busy roadways, and hence, automobile traffic traveling only on these road segments can be measured.

In the recent years, modern technologies such as Global Positioning System (GPS)-enabled smartphones have emerged as a promising alternative to overcome the data collection drawbacks posed by physical sensors. Such location-enabled smartphones can provide continuous streams of mobility data for individual vehicles that can support the construction of their time-varying models. These vehicle-level models can then be fused to form a more complete and nuanced picture of network conditions. An additional advantage of using smartphones is that the measurements are not limited to automobiles but can also be generated as a traveler navigates other modes including mass transit and pedestrian walkways. Therefore, smartphone measurements can support multi-modal models to characterize an individual's behavior traversing multiple means within a

transportation network. These models can thus provide a holistic view of how different modes interact, which have traditionally been studied in isolation. Finally, crowdsourcing data collection to smartphones is virtually free, incurs no deployment and maintenance costs, and promotes the safety of roadway workers.

Despite their promise, there is very little research on the suitability of GPS-enabled smart phones to monitor traffic conditions across the entire transportation network. Existing works demonstrate a narrow scope, such as: (i) measuring traffic from fixed or static locations [4, 2]; (ii) restricting to specialized networks such as highways [1] or university campuses [4]; or (iii) covering only one mode, for example, public transportation [5]. This paper reports on the architecture and non-functional properties of a general-purpose smartphone-based monitoring system to collect continuous streams of traffic measurements. The system captures a sequence of locations and transmits these data points over the public Internet to a server for storage and processing. Experimental deployment, data collection, and analysis on the transportation network around UConn demonstrates the promise of using crowdsourced technology for efficiently collecting richer traffic data through the web of roadways spanning the entire transportation network. Such detailed data can feed sophisticated models which can be used to plan capacity, enhance road safety, and reduce congestion.

The rest of the paper is organized as follows: Section 2 describes the system architecture and its properties. Section 3 presents the preliminary experiment and analysis. Section 4 surveys related research. Section 5 offers conclusions and directions for future research.

2 Traffic Monitoring System

We describe the architecture and non-functional properties of the smartphone system for collecting traffic data.

2.1 Architectural View

The following three major components of the smartphone system are organized as a pipeline. This section describes the functions and the technologies used to implement these components and the communication between them.

The **Smartphone Application** has a simplistic interface, with just two buttons. One button begins and the other one ceases the data collection. Clicking on the “Start Trip” button generates a random numeric identifier, which remains constant through a session. Unlike sensors that measure traffic only from static locations, this identifier will enable a traveler’s path to be re-constructed because of its association with a sequence of data points that contain the traveler’s time and GPS locations.

The application then runs a simple loop that periodically acquires the phone’s longitude and latitude coordinates. The period or the time interval between the sampling of two data

points denoted by *timeStep* is a configuration parameter of the application. A counter is initialized to zero and incremented each time the coordinates are sampled. The numeric values of longitude and latitude are concatenated to two separate comma delimited strings. When the counter reaches a preset threshold denoted by *dataPoints* in the code, the sequence of coordinates is transmitted to a database server through the Internet. After transmitting the current sequence, the application begins collecting a new one. This process repeats indefinitely or until the application is terminated by the user. The number of data points that the application must collect before transmitting the sequence is also a configuration parameter of the application.

We used Sencha Touch [6] and PhoneGap [7] to implement the smartphone application because professional developers at the University of Connecticut (UConn) recommended this combination of technologies, and offered technical support. Sencha Touch is a HTML5 mobile application framework to build apps for iOS, Android, BlackBerry, Windows Phone, and other devices. It exposes the native application programming interfaces (API) of various devices through its unified framework, hoping to achieve a high degree of platform independence. PhoneGap is an open source application container technology to create natively-installed applications for mobile devices with HTML, CSS, and JavaScript. It can output a binary application for deployment to a particular platform such as IPA file (iOS Application Archive) or an APK file (Android Package). Thus, Sencha Touch and PhoneGap support the development and deployment of programs that target multiple platforms with minimal effort.

The **Database Management System (DBMS)** uses MySQL database running on an Apache web server [8] to store the data collected by the smartphone application. The web server is hosted within the Department of Civil & Environmental Engineering at UConn. This server resides behind the School of Engineering (SoE) firewall under the authority of ECS and is password protected to prevent unauthorized access. Both MySQL database and Apache web server are a part of XAMPP [9], which is an open source Apache distribution containing MySQL, PHP (PHP: Hypertext Preprocessor, formerly Personal Home Page) [10], and Perl. MySQL supports database creation and standard structured query language (SQL) commands to dynamically insert, update, and delete data. The Apache Project develops and maintains an open source HTTP server for operating systems, including UNIX and Windows NT.

The SQL table to store traffic data has four fields:

1. **vehicle_ID**: A numeric identifier of 10 digits, generated by the smartphone application and common to all data points in a single trip.
2. **t_stmp**: The timestamp at which a longitude/latitude pair was sampled by the smartphone.
3. **lon**: Longitude of a data point.

4. **lat**: Latitude of a data point.

We note that the smartphone only collects and transmits the latitude and longitude coordinates of the phone. We can convert these latitude and longitude sequences into the distance or displacement between successive points. For example, GeoDataSource (<http://www.geodatasource.com/developers>) provides source code to perform this conversion in 12 different programming and scripting languages. These displacements can be used to compute acceleration and deceleration.

A **PHP Web Page** was implemented and deployed on the server hosting the MySQL database to transmit data from the smartphone application to the DBMS. This PHP web page contains a form and code to process the data submitted through the form. Figure 1 shows the form, which creates a channel for the data as it is transmitted from the smartphone application to the MySQL database. The smartphone appli-

- **vehicleID**
- **timeStamp**
- **timeStep**
- **latSeq**
- **lonSeq**
- **passwd**

Figure 1. PHP Form for Data Channel

cation automatically populates the fields shown in Figure 1 to the form. *vehicleID* is the numeric identifier, *timeStamp* is the time at which the initial data point in a sequence was collected and *timeStep* is the constant interval between the data points. *lonSeq* and *latSeq* are comma delimited strings of signed double precision values representing the longitude and latitudes tuples. *passwd* is a hidden field that does not display on the web page. The password is embedded in the application to ensure that the data is in proper format before it is inserted into the database. This also prevents users from injecting garbage data. The PHP web page processes the data posted through the form only after authentication. It then dynamically constructs an SQL command to insert the sequence into the DBMS. Figure 2 shows how the components of the system communicate during a typical session of the application. Once the application is started, the traveler’s location is sampled periodically and recorded. After collecting a certain number of data points, the application transmits this sequence to the database server via the PHP enabled web page, which builds a sequence of SQL queries that insert the location and time data. The database supports queries for analysis and modeling.

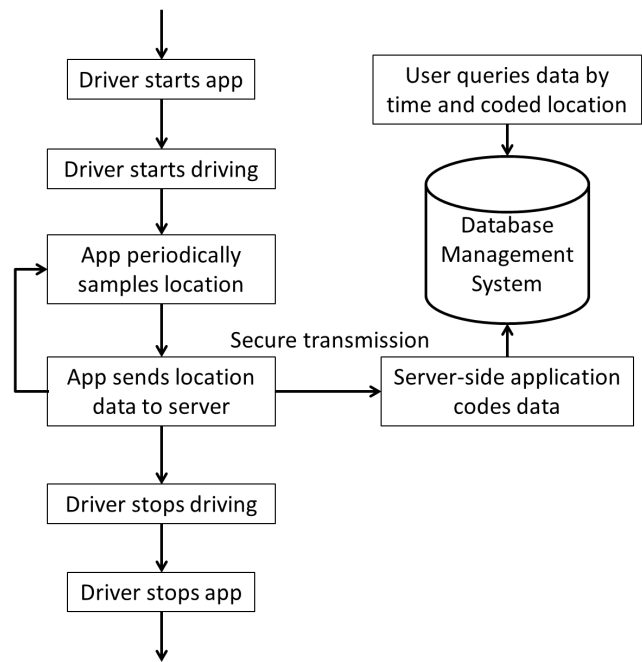


Figure 2. Communication Between Components of Smartphone System

2.2 Architectural Properties

In this section, we outline our design choices to achieve key non-functional attributes.

Performance is concerned with the timely delivery of the data. Sending each latitude-longitude pair as it is sampled, however, will be inefficient and drain a phone’s battery rapidly. On the other hand, transmitting data in batches sacrifices the timeliness of delivery, which can be vital to predict congestion in real time. Thus, performance and energy efficiency conflict, and we balance these two competing goals by grouping a series of measurements and then transmitting this entire sequence.

Transmitting a group of measurements also reduces the number of bits transferred. This is because the application acquires the location data at fixed time intervals, and hence, the timestamps of the entire sequence of coordinates contained in the group can be inferred based on the initial one. Obviously, the savings in bits realized by this choice increases with the number of coordinate tuples grouped together in a sequence. Nevertheless, the size of the sequence transmitted grows with the number of tuples. The extreme case will involve transmitting the entire sequence at once at the end of the trip. However, this deferred delivery will delay the prediction of congestion, and in some cases even make it irrelevant. Moreover, transmitting a large group of data can be unreliable, if bit error rates cause parts or the entire sequence to be lost.

Reliability ensures that the data is delivered to the server without loss or corruption. Software and application fail-

ures, and poor wireless coverage can cause loss of data. Of these, failures of the wireless channel can be tolerated despite intermittent availability, which may often occur in locations such as underground tunnels with weak GPS signals. To prevent data loss from poor coverage, the application buffers the collected sequences on the phone, and then transmits these when the communication is re-established.

The application promotes **Safety** through simplicity of design that renders it non intrusive. It also directly encourages safe driving by explicitly instructing the user to start the application before and discouraging its use while driving. Thus, the application collects and transmits data in the background without active involvement from the user.

Privacy and security is achieved by not collecting any Personally Identifiable Information (PII) such as the username and password or the hardware identifier of the phone. The application captures the dynamics of each trip by an individual, but generates a new identifier per trip. This eliminates the need to collect PII without sacrificing the per-trip details. To ensure the security and privacy of the data during transit, the Advanced Encryption Standard (AES) [11], from the U.S. National Institute of Standards and Technology (NIST), was incorporated through Sencha Touch.

3 Experimental Study

This section details an experiment conducted by deploying the smartphone application on the Android phone of an undergraduate student. With this phone, she rode on the Orange Line bus route around the UConn campus. The random identifier for the trip was 7225472983 and consisted of 156 latitude-longitude tuples. Her trip commenced at 9:32:17 AM at $41^{\circ}81'32.5792''N - 72^{\circ}24'45.6682''W$ and concluded at 9:44:41 AM at $41^{\circ}80'55.7682''N - 72^{\circ}24'86.3101''W$. The size of the sequence or the number of data points grouped into a single transmission was set to one. Thus, the data points were transmitted as they were sampled.

Table 1 lists the absolute time, and the latitude and the longitude coordinates of the first six points collected during this ride. The table also reports the delay between the two points, although this information is not explicitly transmitted and stored in the database.

Table 1. Sample Sequence of Trip Data

Time	Delay	Latitude	Longitude
9:25:41 AM	N/A	41.80562405	-72.24867839
9:25:46 AM	0 : 00 : 05	41.80553366	-72.24848601
9:25:51 AM	0 : 00 : 05	41.80553954	-72.24845334
9:26:01 AM	0 : 00 : 10	41.80556924	-72.24842785
9:26:06 AM	0 : 00 : 05	41.80568999	-72.24822567
9:26:12 AM	0 : 00 : 06	41.80595009	-72.24786536

Figure 3 shows the route with green dots, where each dot denotes a point where data was collected, transmitted and stored at the server. This data collection route closely ap-

proximates the bus route as shown in Figure 4. We note that

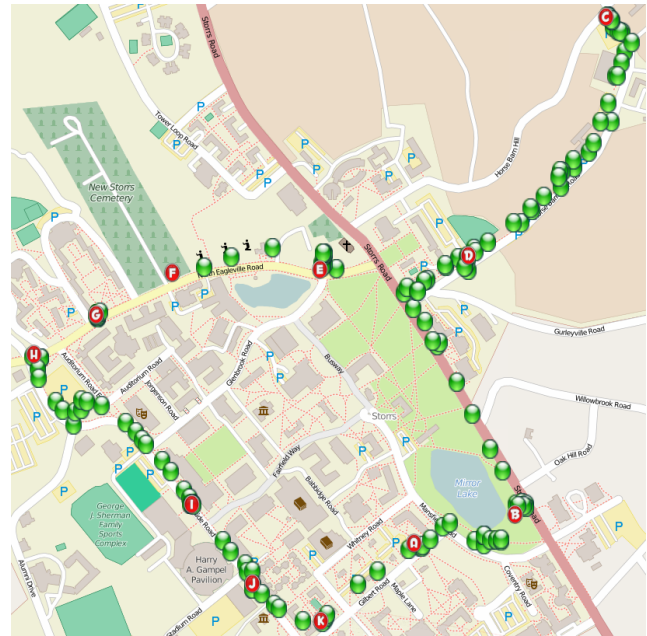


Figure 3. Data Collection Points Along Route



Figure 4. Orange Line Bus Route

the delay between successive coordinates varies slightly during the entire data collection duration. One reason for this variation is weak GPS reception at the phone. For example, the missing data point at 9:25:56 AM in Table 1, may

have been caused by an intermittent loss of signal. Also, the six-second delay between the last two data points in Table 1 may be caused by multiple applications running simultaneously and context switching on the smartphone, leading to non-uniform intervals.

We estimated the speed empirically by computing the distance between two successive geo-coordinates and dividing it by the number of seconds between the two points. The resulting speed is in miles per second, which we convert to miles per hour. We computed a moving average for the i^{th} speed over five points because we found that it was large enough to eliminate some noise, but small enough to prevent excessive smoothing of the data. Excessive smoothing is not desirable because it makes the moving average less responsive to variations, thereby making it harder to detect sudden changes. Figure 5 shows a plot of the moving average of the

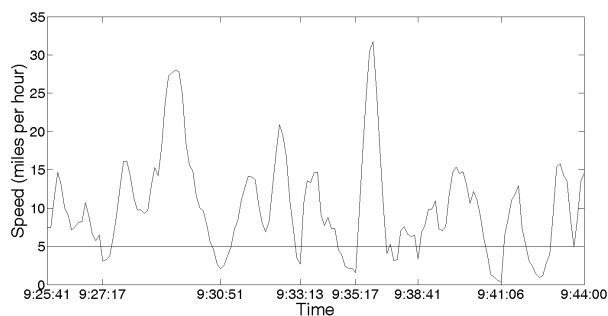


Figure 5. Moving Average of Trip Speed Data

estimated speed of the bus throughout the trip. The straight line in Figure 5 shows a constant speed of five miles per hour for the entire duration of the trip. The moving average of the speed fell below this value nine times. We note that the moving average eliminates speed estimates very close to zero because these are points immediately before a stop and correspond to the bus slowing down. Points immediately after the bus stop correspond to values when the bus was accelerating. We analyzed these local speed minima manually and discovered a strong correlation between the bus stops and the corresponding geo-locations. For example, the trip began at the Arjona stop north of the large lake in Figure 4, marked with a red letter *A* in Figure 3. The stop *B* at 9:27:17 was associated with the Shippee stop. The bus did not stop again until 9:30:51 (*C*) when it reached the Horse Barn Hill Arena at the southern most stop in Figure 4. Subsequent slow downs from 9:33:13 onward include the Young Building (*D*) Inbound stop, the stop light south east of the classics (CLAS) classroom building (*E*), slowing down at cross walk (*F*), a long delay at the North Campus stop at 9:37:34 (*G*), where many students cross the street from dorms to classroom buildings on the other side. The remaining four slow downs from 9:38:41 include the intersection of North Eagleville Road and Hillside Road (*H*) at the top right corner of Figure 4 (left of Figure 3), followed by the Field House (*I*), Co Op (*J*), and Alumni stops (*K*).

4 Related Research

This section compares our work to the conventional and modern approaches to studying transportation networks.

4.1 Conventional Approaches

Conventional approaches that explore transportation data [12] have built tools for examining historical incidents such as accidents. These tools analyze periodically updated archived data and separate data collection from exploration. However, this separation makes it difficult to glean detailed insights on the impact of these incidents on the network dynamics. For example, vehicle crash data often reports time, place, property damage, and fatalities but offers no information on the resulting delay or on how drivers rerouted themselves in response to the incident. In comparison, real-time data collection via smartphones can facilitate greater insights into how traffic changes with time, and how congestion develops and clears. Moreover, the impact of accidents, construction, speed and traffic volume on congestion can be predicted.

State-of-the-art transportation models [13, 14] employ stochastic techniques to estimate the utilization of roads considering travel demand and driver behavior, which is captured in terms of probabilistic selection of routes. Current congestion prediction models use demand data inferred from surveys or fixed sensors. This data includes link volume but not the trends such as acceleration, deceleration, and lane changes that can also indicate congestion. Smartphones can provide richer data to develop detailed models. Because data collection is automated, these models can be validated by collecting data from the same sites. Finally, predictive accuracy of these models can be assessed by collecting data on roads with similar characteristics.

In summary, automated data collection, integrated with exploration and modeling can offer several benefits. First, it can facilitate near real-time monitoring and detection of problems so that alerts can be issued to exercise caution around congestion or an accident and to offer guidance to emergency response officials. Enhanced congestion models can also assess how safety improvements such as new traffic lights impact the roads around the modified location.

4.2 Modern Approaches

Contemporary approaches have used the smartphone technology for the collection of transportation data. Feng *et al.* [1] acquired measurements from GPS-equipped vehicles and developed analytical models to optimally place probe vehicles to minimize the travel time prediction error. Another recent study [2] implemented virtual trip lines, where smartphones collected the location and speed of vehicles as they cross these lines. Lin [3] describes an Android smartphone application called the Toronto Buffalo Border Wait Time (TBBW) to share the waiting time among travelers

on the three Niagara Frontier border crossings. Davami *et al.* [4] describe Kpark, a crowdsourced mobile application to monitoring parking availability on a university campus. Nandan *et al.* [5] identified common challenges in using crowdsourcing for public transportation, and implement an application for demand estimation and next arrival time.

The limitations of these approaches include: (i) studying only specialized transportation networks such as highways [1] and university campuses [4]; (ii) (still) collecting traffic data from fixed, static locations, similar to physical sensors [3, 2]; or (iii) considering only one mode, for example public transportation [5]. Our general-purpose smartphone-based monitoring system, however, can collect dynamic vehicle measurements on a continuum, across the entire web of roadways within a transportation network. It can measure the dynamics of interacting public and private modes including pedestrian walkways and mass transit, facilitating an integrated study.

5 Conclusions and Future Research

This paper describes preliminary results from our efforts to engineer a smartphone-based system to crowdsource the collection of transportation data. An Android smartphone application for gathering user location information was developed and integrated with a server designed to receive, process, and store this data. Experimenting with data collection using the application around UConn campus, and subsequent example analysis of the collected data suggests that our approach can potentially collect richer data sets through the entire transportation network to test the validity of existing models and to develop new ones. Our future work includes enhancements to the application to minimize the involvement of users in data collection. Extensive data gathering experimentation using the application and subsequent analyses are also planned.

Acknowledgments

The authors would like to thank Tiffany and Linda Hoang, Orlando Echevarria and Louis Herman for their help with the smartphone application. This paper was supported by the New England University Transportation Consortium *USDOT/MIT* – 5608530.

References

- [1] W. Feng, A. Bigazzi, S. Kothuri, and R. Bertini, “Freeway sensor spacing and probe vehicle penetration,” *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2178, no. 1, pp. 67–78, 2010.
- [2] J. Herrera, D. Work, R. Herring, X. Ban, Q. Jacobson, and A. Bayen, “Evaluation of traffic data obtained via gps-enabled mobile phones: The mobile century field experiment,” *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 4, pp. 568–583, 2010.
- [3] L. Lin, “Data science application in intelligent transportation systems: An integrative approach for border delay prediction and traffic accident analysis,” PhD, SUNY at Buffalo, Buffalo, NY, 2015.
- [4] E. Davami and G. Sukthankar, “Improving the performance of mobile phone crowdsourcing applications,” in *Proc. of Intl. Conf. on Autonomous Agents and Multiagent Systems*, May 2015 (to appear).
- [5] N. Naveen, A. Pursche, and X. Zhe, “Challenges in crowdsourcing real-time information for public transportation,” in *Proc. of Intl. Conf. on Mobile Data Management*, Jul. 2014, pp. 67–72.
- [6] A. Kumar, *Sencha Touch Cookbook: Over 100 Recipes for Creating HTML5-based Cross-platform Apps for Touch Devices*. Birmingham, UK: Packt Publishing, 2011.
- [7] Adobe Systems, “Phone gap,” *phonegap.com/*, Last accessed March 18, 2015.
- [8] The Apache Software Foundation, “Apache HTTP server project,” *http://httpd.apache.org/*, Last accessed March 18, 2015.
- [9] Apache Friends, “XAMPP Apache + MySQL + PHP + Perl,” *https://www.apachefriends.org/*, Last accessed March 18, 2015.
- [10] A. Tarr and W. Mostrey, *PHP and MySQL 24-hour Trainer*. Indianapolis, IN: Wrox/John Wiley & Sons, 2012.
- [11] J. Daemen and V. Rijmen, *The Design of Rijndael: AES—the Advanced Encryption Standard*. Berlin: Springer, 2002.
- [12] M. Pack, K. Wongsuphasawat, M. VanDaniker, and D. Filippova, “Ice - visual analytics for transportation incident datasets,” in *Proc. of the Intl. Conf. on Information Reuse & Integration*, Las Vegas, NV, Aug 2009, pp. 200–205.
- [13] H. Mahmassani, J. Dong, and B. Park, “Existing traffic prediction and estimation models and systems: Review and summary,” Northwestern University, Tech. Rep. US DOT/FHWA DTFH61-06-D-00005, 2008.
- [14] B. Li, “Recursive estimation of average vehicle time headway using single inductive loop detector data,” *Transportation Research Part B: Methodological*, vol. 46, no. 1, pp. 85–99, 2012.

Endowing NoSQL DBMS with SQL Features Through Standard Call Level Interfaces

Óscar Mortágua Pereira, David Simões, Rui L. Aguiar

Instituto de Telecomunicações
DETI – University of Aveiro
Aveiro, Portugal
{omp, david.simo, ruilaa}@ua.pt

Abstract— To store, update and retrieve data from database management systems (DBMS), software architects use tools, like call-level interfaces (CLI), which provide standard functionalities to interact with DBMS. However, the emerging of NoSQL paradigm, and particularly new NoSQL DBMS providers, lead to situations where some of the standard functionalities provided by CLI are not supported, very often due to their distance from the relational model or due to design constraints. As such, when a system architect needs to evolve, namely from a relational DBMS to a NoSQL DBMS, he must overcome the difficulties conveyed by the features not provided by NoSQL DBMS. Choosing the wrong NoSQL DBMS risks major issues with components requesting non-supported features. This paper focuses on how to deploy features that are not so commonly supported by NoSQL DBMS (like Stored Procedures, Transactions, Save Points and interactions with local memory structures) by implementing them in standard CLI.

Keywords—NoSQL; SQL; databases; middle-ware; call level interfaces; software architecture.

I. INTRODUCTION

Critical data are mostly kept and managed by database management systems (DBMS). To store, update and retrieve data from DBMS, software architects use software tools to ease the development process of business tiers. Among these, we emphasize call-level interfaces (CLI) [1], which provide an API that allows an application to call methods that propagate to the database.

CLI try to build on the commonalities between DBMS and provide a set of methods that encompass these common aspects. Because all DBMS are inherently different, CLI have two main issues to deal with. Firstly, the way of accessing distinct DBMS is different (protocol, format, query language, etc.), which means every DBMS must have its own implementation, which converts the standard API calls to the proper DBMS format. Secondly, DBMS have different features and support different techniques. CLI try to encompass the most common and often seen capabilities, but some DBMS do not support all of them, while others can support features that CLI do not support. Most NoSQL DBMS, for example, do not support transactions, unlike most relational DBMS.

This paper focuses on how to handle this variety of features supported by different DBMS and focusing primarily on features provided by CLI but not supported by the DBMS.

These consist on: 1) transactions, 2) the execution of database functions (like stored procedures) and, finally, 3) interactions with local memory structures, containing data retrieved from the database. We provide a framework that allows a system architect to simulate nonexistent features on the underlying DBMS for client applications to use, transparently to them. It is expected that this research can contribute to minimize the efforts of system architects when DBMS do not support what are considered key features.

The remainder of this paper is organized as follows. Section II presents the state of the art and Section III describes some key functionalities of a CLI (in this case, JDBC). Section IV formalizes our framework, Section V shows our proof of concept and Section VI evaluates our framework. Finally, Section VII presents our conclusions.

II. STATE OF THE ART

There is some work done to bridge the gap between NoSQL and SQL. There have been some solutions focused on providing JDBC drivers to particular DBMS, like [2]–[6], using the DBMS's own query language (usually SQL-like). The authors' approach is to create an incomplete JDBC implementation that delegates CLI requests to the DBMS API and converts the results of queries into JDBC's ResultSet.

There is also work done on translating SQL to the NoSQL paradigm [7]–[12], which allows clients to perform ANSI-SQL commands on NoSQL DBMS. These proposals create a SQL query interface for NoSQL systems, which allow SQL queries to be automatically translated and executed using the underlying API of the data sources.

Work has also been done in an attempt to standardize the access API for NoSQL DBMS. Atzeni et al. [13] propose a common programming interface to NoSQL systems (and also to relational ones) called SOS (Save Our Systems). Its goal is to support application development by hiding the specific details of the various systems. There is also research on cross-database tools that depend heavily on JDBC's features and that cannot be used with NoSQL because their implementations are not complete [14]. To the best of our knowledge, there has not been work done with the goal of implementing CLI's features on drivers and DBMS that do not support them. We expect that our framework positively contributes to overcome the gap between NoSQL and SQL.

III. BACKGROUND

Like previously stated, CLI try to build on the commonalities between DBMS and provide a set of methods that encompass these common aspects. These methods include, for example, reading data from the database, executing commands on it or performing transactions.

Data manipulation commands are usually called ‘CRUD Expressions’, which stand for Create, Read, Update and Delete Expressions, and represent the most common ways to handle data in a DBMS. CLI also usually allow the modification of data on local memory structures, modifications which are propagated to the database transparently, without a client having the need to execute any CRUD expression.

While most full-fledged DBMS have several complete CLI implementations (Microsoft SQL Server, MySQL, among others), some relational DBMS do not (SQLite, for instance) and most NoSQL DBMS do not either.

A. Java Database Connectivity

The Java Database Connectivity (JDBC) [15] is a CLI API for Java. Because of Java’s portable nature, it has been the most popular development language for NoSQL DBMS and, as such, JDBC is the most popular CLI for NoSQL DBMS, even though it is oriented towards relational DBMS.

JDBC Drivers typically return “Connection” objects, which are then used to perform operations on the database. The “Connection” object has a given set of capabilities, which include the creation of CRUD statements to be executed on the database, the creation of statements that call functions inside the DBMS (like Stored Procedures) and the usage of transactions (with commits, roll-backs and save points).

Associated with connections, are ResultSets (RS), which are local memory structures retrieved with "select" queries and representing rows on the database. These use cursors to iterate through their set of data and also allow a set of capabilities, which include retrieval of values from the current row and, if the RS is defined as ‘updatable’, the insertion or deletion of a row and the modification of the current row’s values. These interactions are going to be referred to as ‘Indirect Access Mode (IAM) Interactions’ through the remainder of this paper.

Listing 1 shows the creation of a statement *stmt*, the retrieval of data from table *table1* and how it is kept in the RS (*rs*). Applications are then allowed to update their content. In this case the attribute *attributeName* was updated to *value* and then the modification was committed. We can see how the update is done without the use of any CRUD expression.

```
stmt = conn.createStatement();  
rs = stmt.executeQuery ("select * from table1" );  
rs.update("attributeName", value)  
rs.commit();
```

Listing 1. A query and the update of a value using JDBC.

The features that the driver supports can be further grouped by category: statements (with or without parameters), execution of database functions (stored procedures or user-

defined functions), transactions (and save points), iteration through RS, retrieval of values from RS and IAM interactions.

Some of these features are implemented by all drivers (executing statements on the database, for example). However, the execution of database functions, transactions, save points or IAM interactions is not implemented by some DBMS, depending on their architecture or features. These categories are, then, the focus of this paper.

IV. IMPLEMENTATION FORMALIZATION

To implement these features, there are several options. The first is to create another driver, wrapping the original one, where the methods call the original methods or implement those not supported; the second is to have a server-side middleware layer that intercepts the CLI calls, allows the supported ones and redirects the non-supported ones; the third is to have the clients connecting to the server through a regular socket connection and the server either forwards those requests to a JDBC driver connected to the DBMS or it executes functions from our framework.

While wrapping the driver in another may seem the simplest option (clients can simply use the driver as they usually would, as there is no need for middleware layers to intercept the driver requests or for clients to change the way they connect to the DBMS), it presents some security vulnerabilities, which will be explained further ahead, and also forces the clients to use the modified driver. The second option is the most transparent for clients, but forces a complex implementation on the server, to intercept the JDBC calls and act accordingly, in an imperceptible way for the client. The third option eliminates the need for clients to have any CLI dependency on their code and the server merely acts as a relay from clients to the DBMS. This makes for a simpler implementation of the server logic, but is not transparent to clients. The last two approaches are similar, consisting in a middleware layer able to identify client requests, and any of them are viable. It’s up the each system architect to decide which approach suits his needs the best.

For the remainder of this paper, the middleware layer (intercepting the CLI calls) where the extra logic is implemented will be referred to as the “barrier”. All the client requests must go through the barrier to access the database. It is able to intercept requests and, instead of forwarding them to the DBMS, provide its own implementation and return the appropriate results to the clients, transparently.

A. Execution of Database Functions

A Stored Procedure (SP) is a subroutine available to applications that access a relational DBMS. Typical use for SP include data validation (integrated into the DBMS) or access control mechanisms. Furthermore, they can consolidate and centralize logic that was originally implemented in applications. Extensive or complex processing that requires execution of several SQL statements is moved into stored procedures, and all applications call the procedures. SP are similar to User-Defined Functions (UDF), with a few minor

differences (how many arguments are returned, ability to use try-catch blocks, among others).

If a DBMS does not allow the definition of SP or UDF, these can be implemented on the barrier as a server-side function that calls a group of SQL statements and operations, which are executed together and, therefore, simulate a SP. By doing so, it is possible to simulate most of the behaviors of SP or UDF.

To detect when the functions that simulate SP should be called, there are multiple ways. A simple one would be to give the client the ability to call a SP by the use of a keyword (e.g., *exec storedProcedure1*), where the SP name would be the function name. On the barrier, when the *exec* keyword was detected, a function with the same name as the one requested would be called with the arguments supplied and the results would be returned to the client.

B. Transactions

A transaction symbolizes a unit of work performed within a database, and treated in a coherent and reliable way independent of other transactions. Transactions in a database environment have two main purposes: to provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure; to provide isolation between programs accessing a database concurrently.

A database transaction, by definition, must be atomic, consistent, isolated and durable (ACID). In other words, transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a database must either complete in its entirety or have no effect whatsoever. Furthermore, the system must isolate each transaction from other transactions, results must conform to existing constraints in the database, and transactions that complete successfully must get written to durable storage.

The implementation of transactions is a complex engineering problem, heavily dependent on the DBMS architecture. We present a solution that works with most DBMS, but which also depends on the database schema. Our proposal is defined by, after a transaction has been started, executing the statements in the usual manner, but registering them in a list. If a rollback is ensued, using the list, the changes are undone and return the database to its original state. The implementation of transactions inherently involves the implementation of the ACID properties to a group of statements. Consistency and durability cannot be implemented on the barrier, because these are guaranteed by default by the database itself.

To implement atomicity, along with a list of all the executed actions, there is a need for a list of all the statements that reverse those actions, hereafter referred to as the list of *reversers*. All *inserts* are reversed with a *delete*, all *deletes* with an *insert*, *updates* with *updates* and *selects* do not have to be reverted. To reverse the performed actions, the reverser list of actions must be executed backwards.

One needs to pay attention to the database schema and, if an *insert* triggers other *inserts* (for logging purposes, for example), all of their reversers must be added to the reverser

list. The same happens for cascading *updates* and *deletes*. These kinds of mechanisms are mostly common in relational databases, where transactions are natively supported, so we expect few practical cases where these become relevant.

As an example, imagine a simple transaction consisting of a bank transfer: money is withdrawn from *Account A* and deposited in *Account B*. The money in *A* cannot fall under 0 and the transaction first deposits the money in *B* and then withdraws from *A*. Currently, *A* has 40€, *B* has 0€ and the transaction is executed for a transfer of 50€. When the deposit is made, *B* has 50€ and *A* still has 40€. Here, the increment is registered in the barrier and the reverser (subtracting 50€) is also registered. Then, the transaction tries to withdraw 50€ from *A* but it fails, because the value would go below 0. Here, the transaction is rolled back and the actions in the reverser list would be executed, subtracting the money added to *B* and ending the transaction.

The fact that CRUD expressions are kept on the barrier also has an advantage when implementing transactions. If they were on the client-side, inside the JDBC driver, it would be the client to keep a list of the reversers needed in case of a rollback. If indeed there was a need for a rollback, the client might not have had the permissions to execute those actions and, therefore, could not rollback. To solve this, special permissions would need to be set for this case and that could lead to vulnerabilities that an attacker could take advantage of.

Formally, our definition states that a transaction is composed of actions (which trigger cascading actions), which affect data in the database. Atomicity in a transaction can be implemented if and only if: for any action in any transaction, all the cascading actions can be found; for any action (or cascading action) in any transaction, there is a reverser; the execution of a reverser undoes all and only the changes made by the original action.

Implementing isolation can be done through the use of a single lock (a semaphore or a monitor), which serializes multiple transactions. This concept can be further extended with multiple locks (for example, one for each table), which would allow concurrent transactions if these transactions interacted with (in this example) different tables. Multiple locks can, however, lead to deadlock issues; to avoid them, either one of the transactions has to be reverted (deadlock avoidance/detection) or the locks must all be done at the start of the transaction and must occur in an ordered manner (deadlock prevention).

Because the DBMS does not support transactions natively, reverting one is a heavy process, and it can lead to starvation, depending on which transaction is selected to be rolled-back. The second option, however, decreases the system concurrency and also implies knowing a priori all the tables where changes will be made, which might not be possible.

As an example of the first solution, consider *Transaction A*, which wants to change *Table t1* and *Table t2*; and *Transaction B*, which wants to interact with *Table t2* and *Table t1*, in the opposite order. When the transactions start, both try and lock their first table. Then, one of them, let's say *A*, tries to lock the second table and blocks (because the other transaction, *B*, has

that table locked). When *B* tries to lock its second table, a deadlock situation is detected (because *A* has that table locked) and one of the transactions is rolled back. At that point, the remaining transaction can proceed (because there are no locks on any of the tables now, except its own) and when it is finished, the rolled-back transaction can proceed as well.

As an example of the second solution, consider the same situation. When the transaction starts, both transactions try and lock both tables. To avoid deadlocks, the locks must be done in an ordered manner. In this case, they could be done alphabetically, and not in the order the transactions use them. Both transactions would try to lock *t1* and then *t2*.

The level at which the locks are implemented is also an important choice. With higher levels, implementation is easier, performance is better but concurrency is worse. As an example, imagine a database-level lock. This single lock allows only a single transaction at a time. The cases where such implementation would work in a practical manner are very few. SQLite is one of them, given it is a local file meant to be used by a single process at a time.

Locks at table level, for example, would have better concurrency; clients can perform transactions on different tables at the same time. However, with many clients or very few tables, this level might still be too restrictive. Some NoSQL DBMS may not, however, have the concept of ‘tables’.

Relational DBMS use row-level locks on transactions, which are ideal in the sense that many clients can perform transactions on the same table, just not on the same piece of data they are handling. However, some DBMS may not support row distinction and, inherently, may not support row-level locks. Some NoSQL DBMS also feature millions of rows, which could lead to severe performance issues.

C. Savepoints in Transactions

Assuming transactions have been implemented, the ability to create a save point in a transaction and to roll back to that save point is a simple matter of defining points in the reverser list and only reverting the actions and freeing locks up until that point.

D. IAM Interactions

IAM interactions on a RS consist on the update of values in a row and on the insertion or deletion of rows. By default, a RS’s concurrency type is *read only* and does not allow any of these. If it does, its type is *updatable*. To create a RS that allows IAM interactions, a client must specify it when creating the statement object to execute CRUD expressions on the database.

The barrier can intercept the creation of this statement object and, if the *updatable* type is not supported, wrap the RS that is generated inside our framework’s RS, which simulates the necessary behaviors to allow the insertion, update and deletion of rows. This RS is the one supplied to the client, where he will be able to execute IAM interactions as usual.

Our first approach was the following: when clients attempt to perform actions on the RS (say, inserting a new row), the

actions would be converted and executed like a normal query and the RS would be reset to show the new changes. This had a noticeable performance decay (performing a CRUD expression for the action and another to update the RS) and led to problems when multiple clients were querying the same tables, due to the fact that by resetting the RS, we were re-querying the table fetching results affected by other clients.

Because of this, we followed a different approach where our original RS is never changed (and where we do not have to re-query the table). Values that are updated or inserted are converted to a CRUD expression, inserted in the table and kept in memory. If the client tried to access those values, our framework would present them from memory, without the need to query data from the table. Deleted rows are kept track off and ignored when iterating through the values.

Real Data Structure			
1	A	Deleted	Original RS
2	B		
3	C	Deleted	
4	D		In-Memory Rows
5	E	Inserted + Deleted	
6	F	Inserted	

Client's Perspective on the RS	
1	B
2	D
3	F

Figure 1. Our data structure for IAM interactions with row 2 highlighted.

Figure 1 shows an example of our data structure. When the client requested the RS, rows *A* to *D* were queried. The client inserted *E* and *F* and deleted *A*, *C* and *E*. Rows *E* and *F* are kept in memory, in an array. Rows *A*, *C* and *E* are flagged as deleted. When the client requests the row with index 2, which corresponds to the value *D*, our implementation iterates through the RS, ignoring deleted rows, until we reach the intended row. With this implementation, there is no unnecessary performance decay (there is no need to re-query the data) and there are no concurrency issues (each client can modify their own RS and their inserted/deleted values do not affect the other clients’ RS). This behavior mimics a relational driver implementation’s behavior.

V. PROOF OF CONCEPT

This section describes how the mentioned features were implemented.

A. Execution of Database Functions

To define a SP in a common DBMS, an administrator needs to define four aspects: the name, the input, the output and the actual function of the SP. As such, it is expected that the same aspects must be defined to implement SP on the barrier.

By defining an abstract class *Barrier_CallableStatement* (implementing the *CallableStatement* class), which takes as input a JDBC connection, a name String and an array of arguments (that can be either input or output), the SP framework is defined. To specify the SP, a developer instantiates this abstract class and implements the *execute()* method, which will contain all the SP logic and is the only method that needs to change depending on the SP and the underlying database. As such, all four original aspects are defined and the execution of a SP can be intercepted by the

framework, which will then execute the custom implementation, instead of trying to run it on the database, which would throw an error.

As an example, Listing 2 shows a stored procedure *getEmpName*, defined in MySQL, which returns the name of an employee based on his ID, by querying a table *Employees*, with the fields *id* and *name*.

```
SELECT
CREATE PROCEDURE 'Emp'. 'getEmpName'
(IN EMP_ID INT, OUT EMP_NAME VARCHAR(255))
BEGIN
SELECT name INTO EMP_NAME
FROM Employees WHERE ID = EMP_ID;
END
```

Listing 2. Stored Procedure in MySQL.

The usage of this SP in a Java client with a JDBC connection is shown in Listing 3. A *CallableStatement* is created from the connection object with the SP invocation SQL string. The input and output parameters are defined, the procedure is executed and output parameter is read. We can see that there are two separate definitions of the same procedure, one in the database and one in the client. Because the SP and the barrier are in the same place, this redundant definition should not be needed. When implementing a SP, a developer extends it to the *Barrier CallableStatement* class and defines the number of arguments and the SP name. The execute method contains all the logic (reading input, processing and setting the output).

```
CallableStatement stmt = connection.prepareCall
("call EMP.getEmpName (?,?)");
stmt.setInt(1, employeeID);
stmt.registerOutParameter(2, VARCHAR);
stmt.execute();
employeeName = stmt.getString(2);
```

Listing 3. Invocation of the SP in a Java Client.

The usage of this class is quite similar to the original invocation of the SP and is shown in Listing 4. There is no need to register which parameters are *output* and, in this case, there was no need to refer to the SP name. The barrier, however, keeps a list of the implemented SP and, when it detects a command like *exec getEmpName*, matches the desired SP, executes it and returns the corresponding results.

```
CallableStatement stmt = new SP_getEmpName(conn);
stmt.setInt(1, employeeID);
stmt.execute();
employeeName = stmt.getString(2);
```

Listing 4. Invocation of the SP implementation in a Java Client.

B. Execution of Transactions

Transactions are implemented with an abstract class, just like SPs. Each implementation depended on the underlying DBMS and the methods that must be overridden are the methods that return the reversers. When the execution of a statement is requested, the reverser is determined and the corresponding lock is activated. Then, the statement is executed and the reverser is added to the list of actions in the current transaction. The commit statement releases the locks being used and clears the list of reversers.

In case it is not possible to find the reverser (for example, if the row about to be inserted is not unique and there is no way to delete this specific row, then there is no reverser to be found), an exception is thrown and the statement is not executed. If the statement's execution throws an error, the reverser is not added to the list. A rollback executes all the reversers in the list backwards and clears the list.

If deadlock is detected, one of the transactions is rolled-back. The choice of which transaction is selected can be random, by most recent transaction (first come, first served logic), by which transaction detected the deadlock or by which transaction is easiest to rollback (while better on performance, can lead to starvation). The ease of rollback can be determined by the size of the actions list or, if actions have different impacts, by the calculation of the impact of all the actions currently in the list.

Listing 5 shows an example transaction in a Java client. The database has a table *tb*, on which are inserted two tuples, *A* with ID=1 and *B* with ID=2. The *A* value is committed and therefore, is stored in the database. The *B* value is rolled-back and is not stored in the database. Assuming the table was empty at the start of the transaction, by the end of the transaction, a query should show only a single value, *A*.

```
conn.setAutoCommit(false);
try (Statement stmt = conn.createStatement()) {
stmt.execute("insert into tb values (1, 'A')");
conn.commit();
stmt.execute("insert into tb values (2, 'B')");
conn.rollback(); }
conn.setAutoCommit(true);
```

Listing 5. A simple transaction in a Java client.

As before, a transaction using our framework is expected to function in a similar manner. Listing 6 shows the same transaction, using our framework for SQLite. The creation of the *Barrier Transaction* object matches the setting of *Auto Commit Mode* to *false* in Listing 5 and it handles the creation of the statement object. Then, *A* is inserted and committed, *B* is inserted and rolled-back and the transaction is closed, which matches the setting of *Auto Commit Mode* to *true*.

```
Barrier Transaction trans =
new Barrier TransactionSQLite (conn);
trans.execute("insert into tb values (1, 'A')");
trans.commit();
trans.execute("insert into tb values (2, 'B')");
trans.rollback();
trans.close();
```

Listing 6. A transaction using our Framework.

In a SQL compliant DBMS, when each *insert* action is requested, the corresponding *delete* action is created. For the *A* value, for example, the reverser is *delete from tb where id=1 and name='A'*. On DBMS with different query languages (like Hive), the parsing and creation of reversers would be different. Hence the fact that each DBMS and each schema have its own implementation of the *Barrier Transaction* class; schemas with trigger actions need different implementations from schemas without them.

There is also a need for a client-wide lock system to be deployed to enforce isolation, as well as a system to prevent

deadlocks when handling concurrent transactions. Corbett et al. [16] have shown that there are many different solutions for deadlock detection, both distributed and centralized. In our case, the barrier layer acts as a centralized lock system to guarantee isolation among transactions and, as such, it makes sense to use a centralized deadlock prevention mechanism. We have used table-wide locks with MySQL and Hive and row-level locks with Redis and MongoDB.

When a client performs an action during a transaction, the appropriate reverser is found. Immediately after it is determined, the lock is requested to the *Concurrency Handler* (CH), which requires two things: the URI of the lock (in this case, table names or row keys) and the URI of the requesting process. The CH uses semaphores as locks and creates them as transactions request them. In other words, the first time a client requests the lock for table *t1*, that semaphore is created. Any following requests for that table use that semaphore. This removes the need for our framework to know the database schema and be flexible for any lock-level.

The CH does not lock the semaphore immediately. Before doing so, it checks whether a deadlock situation would be created. It does so by using a graph structure that represents subjects (each transaction) and objects (each table/row) and checking for cycles. If a cycle were to be created by this lock request, that a deadlock situation would emerge [17].

Figure 2 shows an example using the previously mentioned example of transactions *A* and *B* trying to lock tables *T1* and *T2*. We can see that we have a deadlock situation. *B*'s request to *T1* leads to its owner, *A*, which has requested *T2*, which belongs to *B*. In our implementation, this situation would never be reached. Assuming *A* requested *T2* before *B* requested *T1*, when *B* made its request, the cycle would be revealed and the transaction would be restarted. When it rolled-back, its locks would be released, which would allow *A* to proceed. When *A* finished, *B* would be able to lock both tables and execute as well.

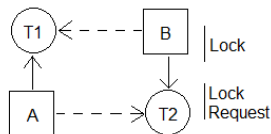


Figure 2. A graph representation of a deadlock situation.

While this example only features two subjects and two objects, the concept can be easily extended for multiple subjects and objects. By solving the deadlock issues, the use of these locks enforces isolation among each transaction. Given that the transactions cannot access another transaction's table/row, then values being read, modified, deleted or created are safe from concurrent modifications.

C. Save Points

A client can set a save point in a transaction and roll-back only up to that save point, which allows for fine-grained control when handling transaction exceptions. Listing 7 shows a transaction that inserts 3 values but only rolls-back one of them (value *B*). Our save point implementation is based in the *Barrier_Transaction* class and, logically, depends on each underlying DBMS. To use save points, a client executes all the

methods, just like previously shown, on the *Barrier_Transaction* object.

```

setAutoCommit(false);
try (Statement stmt = conn.createStatement()) {
    stmt.execute("insert into tb values (1, 'A')");
    conn.setSavepoint("savepoint_one");
    stmt.execute("insert into tb values (2, 'B')");
    conn.rollback("savepoint_one");
    stmt.execute("insert into tb values (3, 'C')");
    conn.commit();
}
conn.setAutoCommit(true);
  
```

Listing 7. A transaction with savepoints in a Java client.

D. IAM Interactions

Interactions on a RS imply that the RS has been requested with the *updatable* type, which enables them. By default, the type is *read only*. Listing 8 shows how a Java client can create a RS, update the third row, insert a new one and delete the second one.

```

Statement stmt = connection.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(
    "SELECT * FROM person");
  
```

Listing 8. A Java client creating a RS to perform IAM interactions.

Because it is our aim to provide as much transparency as possible, the biggest difference is the object request, which uses our wrapper class, as shown in Listing 9. We do not need to specify the type (*updatable* or *read only*) because we assume the DBMS only supports *read only*.

```

ResultSet rs = new Barrier_ResultSetSQLite(
    connection, "SELECT * FROM person",
    ResultSet.TYPE_SCROLL_SENSITIVE);
  
```

Listing 9. A Java client creating a RS with our SQLite implementation.

Our implementation depends on the underlying DBMS, because it depends on the query syntax, as previously stated.

VI. EVALUATION

To demonstrate the soundness of our approach, we have selected four DBMS with different paradigms: SQLite, a relational DBMS; Hive v1.0, a NoSQL DBMS; MongoDB v3.0.2, a document-oriented DBMS; and Redis, one of the most popular key-value DBMS. We expect that our concepts are general enough to be adapted to most NoSQL DBMS. As a basis for comparison, we also used a full-fledged relational DBMS, MySQL, which served as a comparison basis between our barrier implementation and an actual database engine implementation.

The lock-levels were set as tables for all tests, although Redis and MongoDB could use row IDs. Because Redis does not provide a functional and up-to-date JDBC driver, we developed our own driver, which uses the Redis Java API and converts a simple query language into Redis' operations.

The choice of which DBMS to use was done taking into account two main aspects: diversity (it is our goal to show that our concept works with any kind of DBMS, and so it is

important to have both relational and non-relational DBMS, as well as different NoSQL paradigms) and popularity (it is important to choose widely used DBMS).

We tested our framework in a 64-bit Linux Mint 17.1 with an Intel i5-4210U @ 1.70GHz, 8GB of RAM and a Solid State Drive. All the databases were deployed locally, including Hive, which was set-up together with Hadoop as a single-node cluster in this machine. The tests performed include the insertion, update and deletion of values both outside and inside a transaction from our framework.

Op.	Rows	SQLite		MongoDB		Hive	
		Off	On	Off	On	Off	On
Insert	100	749	754	120	189	2642k	2780k
	500	3699	4031	420	1051	X	X
	1000	7907	8494	718	2309	X	X
Update	100	755	758	111	138	3038k	3120k
	500	4025	4096	731	1158	X	X
	1000	8248	8423	2010	3325	X	X
Delete	100	737	746	65	103	2919k	3080k
	500	3648	3784	403	761	X	X
	1000	7502	7775	1123	2018	X	X
Select	100	7	8	81	79	160k	161k
	500	105	107	425	422	X	X
	1000	295	292	1135	1097	X	X

Table 1. A comparison of times taken (in ms) to perform operations in different DBMS with our framework’s transactions enabled and disabled.

Tests (shown in Table 1) show an expected performance decay on all databases. In SQLite, the decay amounts to approximately 8% of the original time taken for the insert operation, 2% for the update operation and 3% for the delete operation. In MongoDB, the decay is much stronger, with over 200% decay for inserts, 60% for updates and 80% for deletes. For Hive, tests could only involve up to 100 rows, due to time restraints. However, Hive shows good results of about 5% decay in inserts, 3% in updates and 5% in deletes. Tests for MySQL and Redis were not considered to have relevant information and were not included.

Because queries are an integral part of the transaction process, the decay is directly related to the ratio between the time taken for queries and operations for each DBMS. This explains why MongoDB has a much stronger decay than SQLite or Hive.

Tests were also conducted in regards to database-stored functions, rollbacks and IAM interactions. The tests show that the performance decay is directly related to the performance of a CRUD expression on the database: if a statement takes 10 seconds, an IAM interaction will also take 10 seconds, plus a residual processing time (about 5 to 10 microseconds). The same relation exists for rollbacks and stored procedures which involve operations in the database.

VII. CONCLUSION

We have proposed a framework that implements some features on a JDBC driver that are not usually implemented using NoSQL drivers. Our proposal includes a model to use our framework in a way that allows concurrent clients to perform atomic and isolated transactions, as well as IAM interactions and database functions, like stored procedures. We have proven our concept with SQLite, Hive, Redis and

MongoDB, and we expect our model to be general enough that it can be extended to other DBMS, relational or NoSQL.

Our performance results show that the use of our framework can be suitable for a real-life scenario. However, work is underway to perform a more in-depth performance evaluation of the different DBMS, with different test conditions, which will be adequate to each DBMS’s architecture and design and provide a more insightful analysis. Work is also underway to add fault tolerance to our proposal; our framework does not currently provide atomicity in case of hardware failures.

In conclusion, our framework positively contributes to overcome the gap between NoSQL and SQL. It helps system architects to simulate key relational DBMS features on NoSQL databases that do not natively support them and eases the transition from a DBMS to another, by abstracting underlying features of the DBMS.

REFERENCES

- [1] ISO/IEC, Information technology -- Database languages -- SQL -- Part 3: Call-Level Interface (SQL/CLI). 2008.
 - [2] R. Öberg, “Neo4j JDBC,” 2015. [Online]. Available: <https://github.com/neo4j-contrib/neo4j-jdbc>. [Accessed: 11-Mar-2015].
 - [3] E. Horowitz, “MongoDB JDBC,” 2010. [Online]. Available: <https://github.com/erh/mongo-jdbc>. [Accessed: 11-Mar-2015].
 - [4] R. Felix, “CouchDB JDBC,” 2009. [Online]. Available: <https://github.com/felix/couchdb-j>.
 - [5] Apache, “HBase JDBC,” 2011. [Online]. Available: <http://www.hbql.com/examples/jdbc.html>.
 - [6] Apache, “Hive JDBC,” 2014. [Online]. Available: <https://cwiki.apache.org/confluence/display/Hive/HiveJDBCInterface>.
 - [7] W.-C. Chung, H.-P. Lin, S.-C. Chen, M.-F. Jiang, and Y.-C. Chung, “JackHare: a framework for SQL to NoSQL translation using MapReduce,” *Autom. Softw. Eng.*, vol. 21, no. 4, pp. 489–508, 2014.
 - [8] R. Vilaça, F. Cruz, J. Pereira, and R. Oliveira, “An effective scalable SQL engine for NoSQL databases,” in *Distributed Applications and Interoperable Systems*, 2013, pp. 155–168.
 - [9] J. Taylor, “Querying a not only structured query language (nosql) database using structured query language (sql) commands.” *Google Patents*, 18-Dec-2013.
 - [10] A. Caillil and R. dos Santos Mello, “SimpleSQL: a relational layer for SimpleDB,” in *Advances in Databases and Information Systems*, 2012, pp. 99–110.
 - [11] R. Lawrence, “Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB,” *Computational Science and Computational Intelligence (CSCI)*, 2014 International Conference on, vol. 1, pp. 285–290, 2014.
 - [12] J. Tatemura, O. Po, W.-P. Hsiung, and H. Hacigümüş, “Partiql: An elastic SQL engine over key-value stores,” in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 629–632.
 - [13] P. Atzeni, F. Bugiotti, and L. Rossi, “Uniform access to non-relational database systems: The SOS platform,” in *Advanced Information Systems Engineering*, 2012, pp. 160–174.
 - [14] B. M. Clapper, “SQLShell.” 2012.
 - [15] Oracle, “JDBC Overview.” [Online]. Available: <http://www.oracle.com/technetwork/java/overview-141217.html>. [Accessed: 09-Mar-2015].
 - [16] C. Corbett, “Evaluating Deadlock Detection Methods for Concurrent Software,” vol. 22, no. 3, 1996.
- M. Singhal, “Deadlock detection in distributed systems,” *Computer*, vol. 22, pp. 37–48, 1989.

Optimization of an Object–Oriented File System

Ling–Hua Chang

Department of Information Management
Kun Shan University, No.195, Kunda Rd., YongKang
Dist., Tainan City 710–03, Taiwan (R.O.C.)
changlh@mail.ksu.edu.tw

Sanjiv Behl

Thomas Edison State College
101 W. State St, Trenton, NJ 08608–1176
sanbehl@yahoo.com

Abstract—Our research provides a unified and coherent presentation of the essential concepts and techniques of object-oriented file systems. It consolidates the results of research and development in the semantics and implementation of a full spectrum of information system facilities for object-oriented systems, including data modeling, querying, storage structures, composite objects and integration of a programming language. This approach presents a tool for building an object-oriented file system called object-oriented file system tool (or OOFS for short) for completing the development of a large object-oriented information system, and its associated applications development framework. First we present the technological objectives underlying the project. Then we present the process of developing the information system and detail its architecture and construction, concentrating on the areas in which object-oriented technology has had a significant role.

Keywords—*object serialization; object data modelling; object-oriented database; information system generator; web system generator*

I. INTRODUCTION

A data model organizes data elements and standardizes how the data elements relate to one another. Object-oriented data modeling has achieved great popularity in recent years. The major factor contributing to its success is that object-oriented data modeling offers its users a high level of abstraction for the representation of information in a manner close to users' conceptual view of that information. We present a tool called OOFS to build an object-oriented file system. This can be used in the development of a large object-oriented information system, and its associated applications development framework. The primary focus for OOFS development is the implementation of a large object-oriented information system.

In our earlier work, we developed a customized software tool for automatically generating a complete Java program based on the values or parameters inputted by the user, called ISG [1] [2] [3]. ISG offers users interface screens for generating an information system and has six transformational functions – *building object-oriented file system (OOFS), linking to the next window, building data processing window, displaying data, previewing a designed window and printing data*. The advantage of ISG is that it uses object serialization mechanism to fill objects with data, which saves CPU execution time. The attributes of an object and its path need to be specified for ISG to translate it to Java code. Thus the program can store and retrieve data efficiently. It can also save time in coding, debugging, testing and implementing an information system.

II. RELATED WORK

Some of the papers regarding how to store Java objects are “Reading Large Volumes of Java Objects from Database” [4], “A Framework for Object–Oriented Data Mining” [5], “A Composite Data Model in Object–Oriented Data Warehousing” [6], “Efficient object serialization in Java” [7], “Object Serialization Support for Object Oriented Java Processor” [8], etc.

In 2000, Raimund K. Ege [4] explores issues in his paper that arise when Java programs access objects stored in databases. They report on their experience with designing and implementing an approach that allows a Java program to pretend that all objects are in main memory, and relieving the Java program from most database housekeeping chores. The architecture is supported by APIs to an actual database: the API can map to an object-oriented database, a relational database via JDBC, or to files using object serialization.

In 2008, Linna Li et al. [5] proposed a system called Escher that is very suitable for describing knowledge for object-oriented data mining. Escher supports a variety of data types and can describe complex data. They also presented a framework for object-oriented data mining, where type information of data and semantic information of data model could be used to guide the data mining process. A specific data mining task, the frequent pattern discovery, is investigated under this framework.

In 1999, Wei–Chou Chen et al. [6] introduced a composite data model in which they proposed to store data in an object-oriented data warehouse. The data warehouse is an information provider that collects necessary data from individual source databases to support the analytical processing of decision-support functions. The data model forms new classes consisting of the attributes listed in the definitions of views and copies necessary class structure from the data source. The query performance of the data warehouse can thus be improved. The corresponding view creation and deletion algorithms were also proposed.

The authors in [7] state that object serialization is the ability to write the complete state of an object to an output stream, so that it can be recreated from the serialized representation at a later time. They also present a number of improvements to the serialization mechanism aimed at decreasing pickle sizes without visible degradation in the serialization performance. Through performance results, they show that it produces

pickles up to 50% smaller without degrading the serialization performance.

Another paper “Object Serialization Support for Object Oriented Java Processor” [8] introduces a functional unit which consists of a serialization and de-serialization unit along with the descriptors and pool to describe the stored serialized objects. This design can enhance the performance of Java based mobile devices which run applications that communicate with other similar applications very often. This design makes use of architectural features of processors.

Java is one of the stable object oriented programming languages which is widely used. In the papers mentioned above, the proposed methods do an efficient serialization in object oriented Java processor or applications. Many of the major projects in industry are developed using Java. A suggestion has also been made to enhance the Java serialization package to reflect this hardware enhancement on the overall performance.

III. OOFS SYSTEM ARCHITECTURE

Let’s focus on aspects of OOFS that are particularly appropriate for its use in an e-Business system.

A. OOFS Standard Model

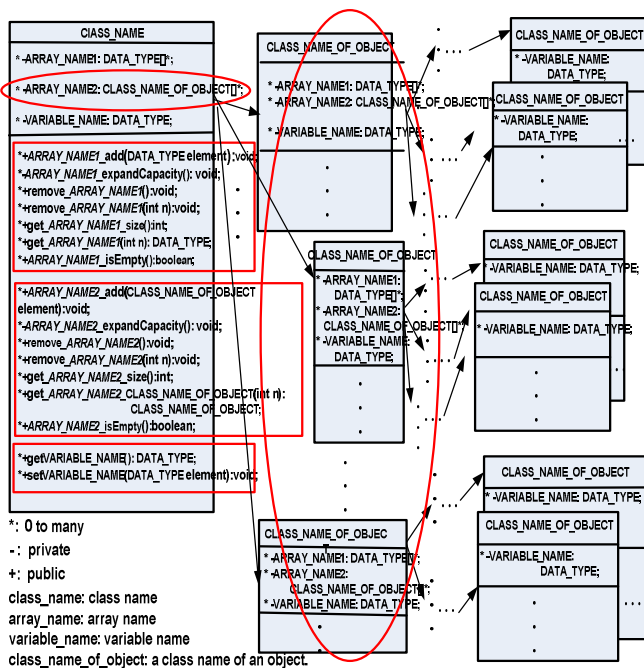


Figure 1. A file structure of OOFS standard model.

OOFS stores data using object streams rather than regular streams because Java has *persistence* in object-oriented circles [9], which means the object layout on disk will be exactly like the object layout in memory. So when an object (the first created object) is saved to disk, the memory addresses of objects that are created subsequently are stored in their associated array and are stored to the disk automatically. Since OOFS introduces this mechanism we design an OOFS standard data model in Fig. 1 to help users to design their file systems.

Fig. 1 shows a file structure of OOFS standard model. Since OOFS builds a file as an object stream file and uses

arrays to manage their associated objects of the same class, a subsequent class is created whose objects are stored in an array of the previous class. Therefore if there are a number of object-oriented arrays in a class diagram, a number of subsequent class diagrams are shown as in Fig. 1 (see two oval marks).

A class diagram with an OOFS design has a class name at the top, its attributes are in the middle and methods which the class can execute are at the bottom. Attributes of OOFS contain variables and arrays.

A variable in OOFS is defined as VARIABLE_NAME: DATA_TYPE. A variable named VARIABLE_NAME can reserve memory locations to store a value. Data type DATA_TYPE of the variable decides what can be stored in the reserved memory. There are 9 data types supported by OOFS including byte, short, int, long, float, double, boolean, char and String.

OOFS provides two types of arrays are ARRAY_NAME1:DATA_TYPE[]* and ARRAY_NAME2: CLASS_NAME_OF_OBJECT[]* (See the first oval mark in Fig. 1). []* means the number of dimensional array. Currently OOFS only works for arrays that have up to three dimensions.

ARRAY_NAME1:DATA_TYPE[]* stores a fixed-size sequential collection of elements of the same data type (DATA_TYPE mentioned has 9 data types) and the array is named ARRAY_NAME1.

ARRAY_NAME2:CLASS_NAME_OF_OBJECT[]* describes array ARRAY_NAME2 created uses defined the constructor of the class named CLASS_NAME_OF_OBJECT which is used to access objects.

For a variable VARIABLE_NAME: DATA_TYPE, OOFS offers two methods to manage variable VARIABLE_NAME and whose methods are getVARIABLE_NAME() and setVARIABLE_NAME(DATA_TYPE element). getVARIABLE_NAME() returns the element stored in this variable VARIABLE_NAME and setVARIABLE_NAME(DATA_TYPE element) sets the element stored in this variable VARIABLE_NAME (See the third rectangular mark in Fig. 1).

For an array ARRAY_NAME1: DATA_TYPE[]*, OOFS offers 7 methods to manage this array and which are ARRAY_NAME1_add(DATA_TYPE element), ARRAY_NAME1_expandCapacity(), remove_ARRAY_NAME1(), get_ARRAY_NAME1_size():int, get_ARRAY_NAME1(int n):DATA_TYPE, ARRAY_NAME1_isEmpty() (see the first rectangular mark in Fig. 1).

For an array ARRAY_NAME2: CLASS_NAME_OF_OBJECT[]*, OOFS also offers 7 methods to manage the array and which are ARRAY_NAME2_add(CLASS_NAME_OF_OBJECT element), ARRAY_NAME2_expandCapacity(), remove_ARRAY_NAME2(), remove_ARRAY_NAME2(int n), get_ARRAY_NAME2_size():int, get_ARRAY_NAME2_CLASS_NAME_OF_OBJECT(int n): CLASS_NAME_OF_OBJECT, ARRAY_NAME2_isEmpty() [10] (see the second rectangular mark in Fig. 1). We take array

ARRAY_NAME2 for example and describe how these methods manage array ARRAY_NAME2.

ARRAY_NAME2_add(CLASS_NAME_OF_OBJECT *element*) adds the specified object *element* whose class name CLASS_NAME_OF_OBJECT to array ARRAY_NAME2.

ARRAY_NAME2_expandCapacity () creates a new array to store the contents of array ARRAY_NAME2 with twice the capacity of the old one.

remove_ARRAY_NAME2() removes all of objects from the array.

remove_ARRAY_NAME2(int *n*) operation consists of making sure the array is not empty and removes the specified object from the array using index *n*.

get_ARRAY_NAME2_size():int returns the number of objects in the array.

get_ARRAY_NAME2_CLASS_NAME_OF_OBJECT(int *n*): CLASS_NAME_OF_OBJECT returns an object whose class name CLASS_NAME_OF_OBJECT using index *n*.

The following uses as an example the e-Business system of the Eastland Company to describe these 7 methods in detail.

B. Applying Oofs Standard Model on Eastland e-Business System

Since Oofs is to build the file system of an information system and for linking to the next window and for building data processing window of ISG are to generate graphic user interface screens which are to input data and then store data. Now take Eastland e-Business for example to describe how we implement the file system of Eastland e-Business.

inventory statistics, goods expenses, types of expenses, monthly earning, profit etc. Then use Oofs standard model in Fig. 1 to illustrate the file structure of these 17 groups of classes. Now we document these designs using Oofs standard model to examine three of these groups, for invoice, goods expenses and goods shown in Fig. 2. It shows class diagrams of file INVOICE, file PAYOUT, file GOODS_PAYOUT_KIND. Each of files has its associated classes such as class INVOICE, class PROFORMA_INVOICE_DATA and class PI_DATA in file INVOICE, class PAYOUT and class PAYOUT_DATA in file PAYOUT, class GOODS_PAYOUT_KIND and class GOODS_PAYOUT_KIND_DATA in file GOODS_PAYOUT_KIND.

Taking file INVOICE for example (see the first oval mark in Fig. 2), there are two arrays in object of class INVOICE; array PI_PROFORMA_INVOICE is to store pro forma invoice and array SI_SHIPPING_INVOICE is to store shipping notice and both store objects of class PROFORMA_INVOICE_DATA. Class PROFORMA_INVOICE_DATA includes attributes – PI_NUMBER, PI_COMPANY, PI_CUSTOMER, etc. and an array PI_ARRAY which stores objects of class PI_DATA. Class PI_DATA includes attributes – PI_GU_NUMBER, PI_GU_IDX, PI_GU_STATEMENT, PI_AMOUNT, PI_UNIT, PI_PRICE and PI_PRESENTLY.

Another issue to consider is how an array manages its associated objects. Consider arrays PI_PROFORMA_INVOICE and SI_SHIPPING_INVOICE (see four rectangular marks at class INVOICE in Fig. 2) for illustrating how the object streams are stored in a file. Take array PI_PROFORMA_INVOICE for example and array PI_PROFORMA_INVOICE is used to manage objects of class PROFORMA_INVOICE_DATA and seven methods enclosed in the third rectangular mark are generated by Oofs that are methods PI_PROFORMA_INVOICE_add(PROFORMA_INVOICE_DATA *element*), remove_PI_PROFORMA_INVOICE(int *n*), get_PI_PROFORMA_INVOICE_size(), PI_PROFORMA_INVOICE_isEmpty(), get_PI_PROFORMA_INVOICE_PROFORMA_INVOICE_DATA(int *n*) and PI_PROFORMA_INVOICE_expandCapacity().

If get_PI_PROFORMA_INVOICE_PROFORMA_INVOICE_DATA(int *n*) returns an object of class PROFORMA_INVOICE_DATA named A_PROFORMA_INVOICE1 and using this object to call method getPI_NUMBER() gets value of attribute PI_NUMBER. Therefore ISG can translate getting value of PI_NUMBER with A_PROFORMA_INVOICE1 into a statement A_PROFORMA_INVOICE1.getPI_NUMBER(). Another statement A_PROFORMA_INVOICE1.setPI_NUMBER(*n*) is to set a value *n* to attribute PI_NUMBER. When array PI_PROFORMA_INVOICE is created (means object of class INVOICE is created and array PI_PROFORMA_INVOICE in it), it is allocated a specific number of cells into which elements can be stored. Since we use a fixed-size data structure, at some point the array may become full. So method PI_PROFORMA_INVOICE_expandCapacity() will be called automatically to double the size of array PI_PROFORMA_INVOICE. Thus the file structure of

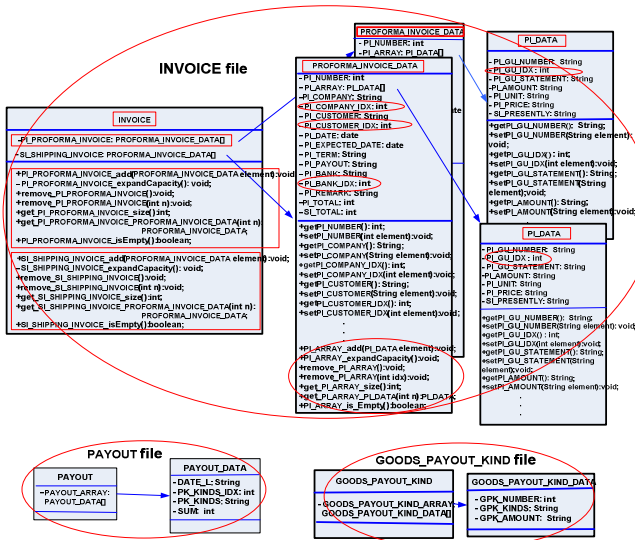


Figure 2. Class diagrams of six groups of classes from Eastland e-Business.

Consider Eastland e-Business that computes the monthly shipping amount, generates reports on their monthly earnings, profit, orders, etc. Let's focus on building the file system of Eastland e-Business and there are 17 groups of classes (classes that are related to each other through composition) in this system such as classes for foreign bank data, local bank data, invoice (including pro forma invoice, shipping notice), company data, product description, vendor purchase orders, products, single

Eastland e-Business, with arrays and methods provided makes it convenient and efficient to retrieve data elements in any array. How users use OOFS to set parameters and translate to Java programs is discussed in [1].

C. How ISG Retrieves data from OOFS File Structure

If users specify a path for retrieving PI_GU_STATEMENT data as INVOICE/PI_PROFORMA_INVOICE/PI_ARRAY/PI_GU_STATEMENT, then ISG uses this path to translate a Java program for getting PI_GU_STATEMENT. INVOICE/PI_PROFORMA_INVOICE/PI_ARRAY/PI_PRICE means that there are two arrays – PI_PROFORMA_INVOICE and PI_ARRAY and there are objects stored in their associated arrays.

The path indicates that an object of class INVOICE contains a PI_PROFORMA_INVOICE array. Each element of PI_PROFORMA_INVOICE array is an object of class PROFORMA_INVOICE_DATA. Each of these objects contains a PI_ARRAY array. Each element of PI_ARRAY array is an object of class PI_DATA. Each of these objects contains an attribute called PI_GU_STATEMENT. Therefore ISG knows how to get attribute PI_GU_STATEMENT by following steps using the above path specified. The command statements can be seen in [2].

1. Read an object named THE_INVOICE from file INVOICE, since PI_PROFROMAN_INVOICE is an array stored in the THE_INVOICE object.
2. Next ISG uses for loop to retrieve each object of class PROFORMA_INVOICE_DATA from the PI_PROFORMA_INVOICE array which is temporarily stored in AN_INVOICE object.
3. There is an array PI_ARRAY in object AN_INVOICE and ISG use this object to get each object of class PI_DATA stored in array PI_ARRAY. ISG also uses for loop to get each object of class PI_DATA as step 2 does.
4. From the retrieved objects of class PI_DATA, we can get the value of PI_GU_STATEMENT by using each of them.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

We analyze the efficiency obtained in terms of the OOFS CPU time using the two tools (ISG and DWL).

A. Analyzing Efficiency of OOFS CPU Time Using ISG

ISG generated an e-Business system for the Eastland Company by generating a total of 74 Java programs. There were 15 user interface screens for entering the shipping cost, monthly expenses, foreign manufacture’s data, local manufacture’s data, foreign bank data, local bank data, packaging data, about products, about the company, foreign customer shipments etc. The average time it took for translating these GUI screens was 14.46 milliseconds (not including I/O time) which is pretty good since ISG moves the parameters to the memory and uses fast access methods. For storing these parameters there are two files—HF file and FRAME file. An HF file is for storing data architecture such as field name, data type, array dimension, size of array and data type of array. A FRAME file is for storing graphic interface screen component and data path. Since OOFS moves everything to memory at once, it reduces the amount of time it takes to read a FRAME

file (it took 288.93 milliseconds to move data). It took just 22.8 milliseconds for reading the HF file. So the total time for translating a screen was 326.19 milliseconds.

B. Analyzing Efficiency of OOFS CPU Time Using DWL

We used DWL [11] to implement an E-commerce web-based system that we call E-POLEMONG for POLEMONG Plastic Company. POLEMONG Plastic Company manufactures eleven types of products viz. telecommunication parts, auto parts, sports equipment, daily supplies, appliances, aquarium supplies, etc. E-POLEMONG displays six items viz. News, Products, About POLEMONG Company, Investor Information, Product Quality and Contact us. We translated 22 web pages using DWL for E-POLEMONG such as News and eleven different types of products like About POLEMONG, Product Quality, Investor Information, Contact Us, etc. We also applied OOFS to retrieve data and then to translate it to web-pages. The average translation time for these web pages was 545.5 milliseconds, with the shortest time being 330 milliseconds and the longest time being around 700 milliseconds.

ISG’s and DWL’s experimental results show that OOFS is scalable. Since our extents are implemented as one object in one segment, as the number of objects in the extension increased, the size of the extent object increased and that space is big enough to keep all the class extension objects needed for a transaction.

C. Another Advantage of OOFS – Join Approach

Note that ISG uses arrays and methods to create an information system that is convenient and efficient for retrieving data elements in any array. Another characteristic of this OOFS standard model is when the drop-down lists of a GUI screen has data items from other files, ISG can combine data from these files into one file. This mechanism is similar to SQL joins in a database. Since an index is an integer pointer (into an array), using this index can identify elements of the array. Therefore we can use the selected indices from a drop-down list to retrieve its associated data elements from the source file. This mechanism for drop-down lists combines data elements from two or more files into a file which is a Joined Approach. Therefore a Joined Approach can eliminate duplication of information when objects may have one-to-many relationships.

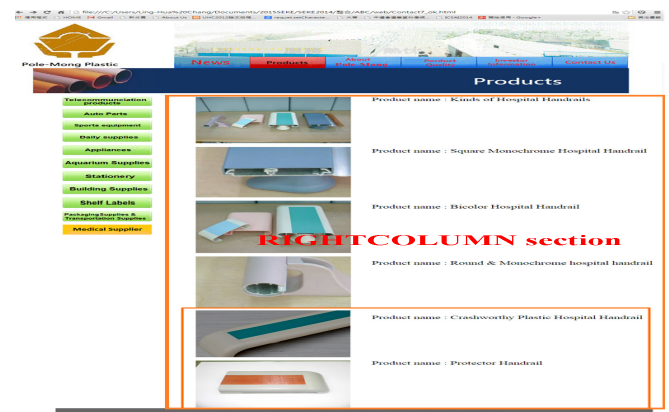


Figure 3: The Medical Supplier page of E-POLEMONG.

D. How to Update Web Pages Quickly as the Company Produces New Products Frequently

Figure 3 shows the Medical Supplier webpage of E-POLEMONG. It shows six types of hospital handrail image pictures. We describe the steps below on how to implement an application that translates it into a new webpage whenever a new product is introduced.

1. Execute *building data processing window* of ISG to generate an interface screen to input data – product name, its image file name, and data is stored into a file called MEDICAL_SUPPLIERFILE.
2. Use DWL to generate an HTML program called MEDICAL_SUPPLIER.html which shows a webpage similar to the one shown in Fig. 3. We wrote a translator to update Fig. 3 which calls PrintWriter to print formatted representations of objects to a text-output stream such as MEDICAL_SUPPLIER.html. There is a for loop in the translator which can be used to make changes to the RIGHTCOLUMN section in Fig. 3. It gets the number of hospital handrails from the MEDICAL_SUPPLIERFILE file. The command statements can be referenced from [11].

V. Conclusions and Future Work

An information system was developed using ISG for Eastland that involves computing the monthly shipping amount, generating reports on their monthly earnings, profit, orders etc. We had already established the usefulness of ISG in our earlier work.

It is convenient and easy to use DWL to generate a web-based system for any company or business. The web-based system enables customers to better understand the company and its products, which would result in increased sales. We illustrated this for a company in the paper. In the future, we hope to use this tool to develop customized web-based systems for other small and medium sized businesses. Both ISG and DWL can save time in writing, debugging and testing a program.

We established the usefulness of DWL in our earlier work which reduced the cost of producing software written in HTML. For example, using DWL we developed a customized web-based system for GREATYO sunglasses for sports and kids [11] and for the 7th Ubiquitous-Home Conference UHC2013 [3]. ISG has been used to generate an e-Business system for Eastland International Company [2]. Our research also offered a tool called W-Revised for creating customized websites. Because companies introduce new products frequently and the web pages of its site need to be updated frequently, it can be done conveniently using W-Revised generated by ISG and DWL [11].

Websites are often hacked by hackers seeking to compromise the corporate network. Also programs are

sometimes downloaded without the users consent or knowledge when they visit a web site (drive-by download). As a result, industry is paying increased attention to the security of the web applications themselves in addition to the security of the underlying computer network and operating systems. In order to keep a website clean and secure, we are trying to find a good solution to defend the websites. Therefore we hope that in the near future our software tools might be included in business applications as software as a service (SaaS).

ACKNOWLEDGMENT

Eastland e-Business System was developed in collaboration with Eastland International Company for implementing an e-Business System. E-POLEMONG was developed in collaboration with POLEMONG Plastic Company for developing a web-based system for their business. We would like to thank them for giving us this opportunity to work with them and test our tools for generating the information and the web-based systems.

REFERENCES

- [1] Ling-Hua Chang, Sanjiv Behl, "An Efficient Information System Generator," 4th Asian Conference on Intelligent Information and Database Systems, pp. 286–297, 2012.
- [2] Ling-Hua Chang, Sanjiv Behl, Tung-Ho Shieh, "Amazing Use of ISG for Implementing Information Systems," 2014 International Conference on Information Science, Electronics and Electrical Engineering, (iseee2014), pp. 1980–1985, April 26–28, 2014.
- [3] Ling-Hua Chang, Tung-Ho Shieh, Sanjiv Behl, "Amazing of Using ISG on Implement a Web-Based System," 14th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'13), pp. 44–49, 2013.
- [4] Raimund K. Ege, "Reading Large Volumes of Java Objects from Database," TOOLS '00 Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00), pp. 117–124, Aug. 2000.
- [5] Linna Li, Bingru Yang, Faguo Zhou, "A Framework for Object-Oriented Data Mining," the 5th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2008), pp. 60–64, Oct. 2008.
- [6] Wei-Chou Chen, Tzung-Pei Hong and Wei-Yang Lin, "A Composite Data Model in Object-Oriented Data Warehousing," TOOLS '99 Proceedings of the 31st International Conference on Technology of Object-Oriented Language and Systems, pp. 400–405, 1999.
- [7] L. Opyrchal, A. Prakash, "Efficient object serialization in Java," Proceedings of 19th IEEE International Conference on Distributed Computing Systems Workshops on Electronic Commerce and Web-based Applications., pp. 96–101, 31 May 1999–04 Jun 1999.
- [8] Joe Cheri Ross, Dr. Priya Chandran, "Object Serialization Support for Object Oriented Java Processor," IEEE Transactions of Information Technology, pp. 1–6, vol. 3 Aug. 2008.
- [9] C.S. Horstmann, G. Cornell, Core Java Volume I-Fundamentals, 8th ed. Sun Microsystems Press, Prentice Hall, New Jersey, 2008.
- [10] John Lewis, Joseph Chase., "Java Software Structures designing and using data structures," Pearson Education Inc., 2005.
- [11] Ling-Hua Chang, Sanjiv Behl, Tung-Ho Shieh, "W-Revised: an Amazing Tool for Creating Customized Websites", 2014 IEEE, DOI 10.1109/ICSAI.2014.7009333, pp.465–470, 2014.

An Evolution Mechanism for Dynamic Physical Applications in the Internet of Things

Kaibin Xie^{1,2}, Haiming Chen¹, Dong Li¹ and Li Cui¹

¹Institute of Computing Technology, Chinese Academy of Sciences, Beijing

²University of Chinese Academy of Sciences, Beijing

Email: xiekaibin@ict.ac.cn, chenhaiming@ict.ac.cn, lidong@ict.ac.cn and lcui@ict.ac.cn

Abstract—With rapid development of the Internet of Things, more and more smart devices are deployed in the physical space. A physical application is composed by several smart devices which provide physical data. The physical applications need appropriate physical information processing systems to process the related data. However, the physical applications are dynamic because of the ever-changing demands in the IoT. So it is necessary to design an evolution mechanism for the dynamic physical applications to find appropriate physical information processing systems. We first analyze the changing types of dynamic physical applications. Then we conclude three relationships between the dynamic physical applications and physical information processing systems. In order to verify the correctness of the evolution mechanism, we use Communication Sequential Process to formalize the evolution mechanism and use Process Analysis Toolkit to verify deadlock-free, divergence-free and nonterminating of the evolution mechanism.

Keywords—Internet of Things; physical application; physical information processing system; dynamic; evolution mechanism

I. INTRODUCTION

The concept of the Internet of Things (IoT) is proposed by MIT in 1999 [6] and has got extensive attention from the industrial community [3]. According to the vision for the IoT [7], most smart devices in the physical space have the ability to communicate and compute. A physical application is composed by several smart devices which provide physical data to the social. These physical applications need appropriate physical information processing systems to process the related data.

In our previous work, we have established a software architecture of the IoT, named PMDA [9]. The PMDA is composed by three models which are extracted from the social space, the virtual space and the physical space. The three models are the Application Model, the Sense-Execute Model and the Physical Model. The relationship of the three models is illustrated in Fig. 1. As depicted in Fig. 1, the Application Model sends requirement information (req-info) to the Sense-Execute Model; the Sense-Execute Model processes sensory data (sen-data) from the Physical Model according to the req-info from the Application Model and sends the execution information (exe-info) to control the Physical Model; the Physical Model provides sen-data to the Sense-Execute Model and receives the exe-info from the Sense-Execute Model.

According to the PMDA, we can see that these physical applications can be regard as the instances of the Physical Model; the physical information processing systems can be regard as the instances of the Sense-Execute Model.

Because of the ever-changing demands in the IoT [10], the physical space and the physical parameters of physical applications are changeable. The changeable physical applications need appropriate information processing systems. In order to adapt the ever-changing demands in the IoT, we design an evolution mechanism for the dynamic physical applications to find appropriate physical information processing systems.

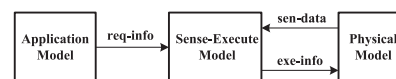


Fig. 1. The relationship of the three models in the PMDA

The challenges for designing the evolution mechanism are as follows. Firstly, it is difficult to find an appropriate physical information processing system for a dynamic physical application in the IoT. Secondly, it is hard to illustrate the correctness of the evolution mechanism. Without strict proof, we can not state that the evolution mechanism is correct in all situations.

The remainder of this article is organized as follows. We provide the related work and a motivational scenario of the evolution mechanism in Section II. Section III analyzes the dynamic changes of physical applications. Three relationships between the dynamic changes of these physical applications and the physical information processing systems are depicted in Section IV. Section V establishes the procedures of the evolution mechanism. The processes of evolution mechanism are depicted by the Communication Sequential Process(CSP) [5] statements and the correctness of the evolution mechanism is verified by the Process Analysis Toolkit(PAT) [4] in Section VI. Finally, we make a concluding remark of the evolution mechanism.

II. RELATED WORK AND MOTIVATIONAL SCENARIO

The evolution for the IoT has been investigated mainly in three aspects which are the changing context and demands for an IoT application [2], the user mobility and unreliable sensor availability in IoT [1] and the dynamic interactions in the IoT [8].

Based on the software architecture PMDA and recent research in IoT evolution, this article analyzes the evolution mechanism for the dynamically changing physical applications due to the ever-changing demands in the IoT. The evolution mechanism can guarantee that these physical applications

evolve correctly according to the physical information processing systems in the IoT. We depict a typical scenario for these dynamic physical applications as follows.

Jim is a supervisor of a large environment monitoring system. The organization deploys environmental monitoring applications in three areas (area A, area B and area C) of a city. Jim deploys the sensors in the three areas in order to provide physical data of the environment to the corresponding physical information processing systems which can process the physical data according to the requirements (Req-A, Req-B and Req-C) from the social space. In area A, Jim deploys temperature sensors and humidity sensors; in area B, Jim deploys CO sensors and CO₂ sensors; in area C, Jim deploys CO sensors and temperature sensors. The three environmental monitoring applications are instances of the Application Model and named as pma, pmb and pmc respectively. Jim develops three physical information processing systems to process the three environmental monitoring applications. The three physical information processing systems are instances of the Sense-Execute Model and named as sema, semb and semc respectively. Fig. 2 shows the scenario of the three environmental monitoring applications.

Jim wants to manage the city's environmental monitoring applications in an effective way even when the environmental monitoring applications have changed. But the above environmental monitoring applications don't conform to Jim's expectations because the three physical information processing systems can not adapt to the dynamic changes in the three environmental monitoring applications.

So Jim asks the Research department to realize the intended environmental monitoring applications. The Research department reports that they should design an evolution mechanism for these environmental monitoring applications. The evolution mechanism can adapt to the dynamic changes in the three environmental monitoring applications and guarantee that there are appropriate physical information processing systems for the three environmental monitoring applications.

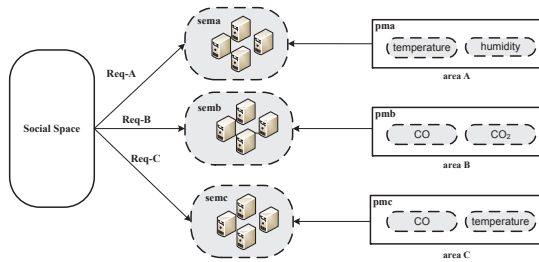


Fig. 2. The scenario of the three environmental monitoring applications

III. CHANGING TYPES OF PHYSICAL APPLICATIONS

Physical location and the physical parameters are two key characteristics for the physical applications. The structure of a physical application is illustrated in Fig. 3. In Fig. 3, the pa represents the name of a physical application; the pl represents the physical location of a physical application and the pps represents the physical parameters of a physical application.

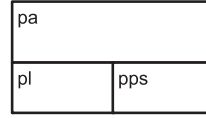


Fig. 3. The structure of a physical application

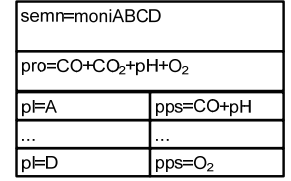


Fig. 4. The structure of a physical information processing system

We take the scenario in section II as an example and the possible changing types of the dynamic physical applications are as follows.

A. Changing types for the pl

There are four changing types which are changed in the pl and remain unchanged in the pps. The four changing types can be depicted as follows.

- 1) Shrink the scope of the physical location in the physical application. The reduced sites are denoted by S. We use SHRINK to represent this changing type.
- 2) Enlarge the scope of the physical location in the physical application. The added sites are denoted by E. We use ENLARGE to represent this changing type.
- 3) Shrink and then enlarge the scope of the physical location in the physical application or the versa. We use SHR-ENL to represent this changing type.
- 4) Move to a new physical location. We use MOVE to represent this changing type.

B. Changing types for the pps

There are four changing types for the pps, which are changed in the pps and remain unchanged in the pl.

- 1) Add new physical parameters. We use ADD to represent this changing type.
- 2) Delete physical parameters. We use DELETE to represent this changing type.
- 3) Delete the physical parameters and then add new physical parameters or the versa. We use DEL-ADD to represent this changing type.
- 4) New physical parameters. We use NEW to represent this changing type.

C. Changing types for both the pl and the pps

Because there are four changing types for the pl and four changing types for the pps, we can conclude that there are sixteen changing types for the changes in both the pl and the pps of a physical application.

D. ZERO changing type

There is a special changing type for the physical application, which denotes that the pl or the pps of a physical application is null. We use ZERO to represent this changing type. The ZERO denotes that the physical application has terminated in the IoT.

IV. RELATIONSHIPS BETWEEN DYNAMIC PHYSICAL APPLICATIONS AND PHYSICAL INFORMATION PROCESSING SYSTEMS

According to these changing types discussed in section III, we analyze the relationships between the dynamic physical applications and the physical information processing systems.

A physical information processing system processes the physical data from the corresponding physical applications according to the requirements from the social space. The process ability of a physical information processing system is denoted by pro. The pro is composed by several physical parameters which the physical information processing system can process in the IoT. A physical information processing system can process these physical applications if the pps of these physical applications are contained by the pro. A physical information processing system records the related pl and pps of the physical applications. We use semn to represent the name of a physical information processing system.

The structure of a physical information processing system is illustrated in Fig. 4.

We conclude that there are three relationships which are named as ERASE, UPDATE and LOOKUP and analyze the three relationships as follows.

A. The ERASE relationship

The ERASE relationship denotes that the changed physical application has nothing with any physical information processing systems in the IoT. We can conclude that the physical application has terminated in the IoT. The changing type for the ERASE relationship is ZERO.

B. The UPDATE relationship

If a physical application has changed and the changed pps for the physical application still contains in the pro of the corresponding physical information processing system, we name it as UPDATE relationship for the physical application.

C. The LOOKUP relationship

If a physical application has changed and the changed pps for the physical application is not in the pro of the corresponding physical processing system, we name it as LOOKUP relationship for the physical application.

V. EVOLUTION PROCEDURES

According to the procedures of the three relationships, we can conclude that there are seven procedures for the evolution mechanism. We illustrate the seven procedures as follows.

- 1) IRS (Initial Relationship Set): The IRS denotes the relationship between the physical applications and the physical information processing systems at initial.
- 2) JUDGE: We judge the changing types for these dynamic physical applications in the IoT.
- 3) ERASE: The relationship is ERASE. After the procedure for ERASE, goto 7).
- 4) UPDATE: The relationship is UPDATE. After the procedure for UPDATE, goto 7)

- 5) LOOKUP: The relationship is LOOKUP. If we can find an appropriate physical information processing system for the changed physical application, goto 7). Else, it denotes that there is no physical information processing system for the changed physical application, goto 6).
- 6) DEPLOY: We deploy a new physical information processing system for the changed physical application and establish the relationship between the changed physical application and the new physical information processing system. After the procedure for DEPLOY, goto 7).
- 7) UPDIRS: We update the IRS in order to form a new relationship between the changed physical applications and the physical information processing systems. After the procedure for UPDIRS, goto 1).

The seven procedures of the evolution mechanism are illustrated in Fig. 5.

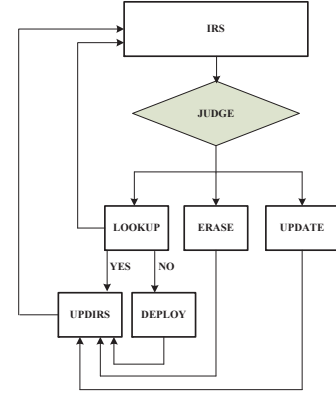


Fig. 5. Procedures of the evolution mechanism

VI. CORRECTNESS VERIFICATION

In order to verify the correctness of the procedures for the evolution mechanism, we express the seven procedures by the processes in the CSP. We verify the correctness of these processes by the PAT.

A. Processes for the evolution mechanism

We use the process IRS to express the procedure 1) in the evolution mechanism. There is only one event (generate) in the IRS. The meaning of the event is to generate the Initial Relationship Set.

We use the process JUDGE to express the procedure 2). There are two events (change and judge) in the process JUDGE. Event “change” shows that the physical application has changed. Event “judge” is to judge the relationship.

We use process ERASE to express the procedure 3). There are two events (erase, unlinkera) in the process ERASE. Event “erase” shows that the physical information processing system deletes the related pl and pps of the physical application. Event “unlinkera” shows that the physical information processing system unlinks with the physical application.

We use the process UPDATE to express the procedure 4). There is only one event (update) in the process UPDATE. Event “update” shows that the physical information processing system updates the pl and the pps.

We use process LOOKUP to express the procedure 5). There are three events (delup, unlinkup and search) in the process UPDATE. Event “delup” shows that the physical information processing system deletes the pl and the pps of the related physical application. Event “unlinkup” shows that the physical information processing system unlinks with the physical application. Event “search” shows that the physical application searches the appropriate physical information processing system in the IoT.

We use process DEPLOY to express the procedure 6). There are two events (link, register) in the process DEPLOY. Event “link” shows that the physical application links to a new physical information processing system. Event “register” shows that the pl and pps of the physical application are registered in the new physical information processing system.

We use four processes (ERAIRS, UPDIRS, LOOKIRS and DEPIRS) to express the procedure 7).

The process ERAIRS updates the IRS in procedure 7) after the process ERASE. The process UPDIRS updates the IRS in procedure 7) after the process UPDATE. The process LOOKIRS updates the IRS in procedure 7) after the process LOOKUP. The process DEPIRS updates the IRS in procedure 7) after the process DEPLOY.

There is only one event “updera” in the process ERAIRS. Event “updera” updates the IRS by deleting the link relationship.

There is only one event “updup” in the process UPDIRS. Event “updup” updates the IRS by updating the pl and pps of the physical application in the physical information processing system.

There is only one event “updlook” in the process LOOKIRS. Event “updlook” updates the IRS by adding the pl and pps of the physical application to the physical information processing system.

There is only one event “upddep” in the process DEPIRS. Event “upddep” updates the IRS by adding the pl and pps of the physical application to the new physical information processing system.

Based on the above analysis of the processes and events we can get ten CSP processes for the procedures of the evolution mechanism as follows.

- $IRS = generate \rightarrow JUDGE;$
- $JUDGE = change \rightarrow judge \rightarrow (ERASE[*]UPDATE[*]LOOKUP);$
- $ERASE = erase \rightarrow unlinkera \rightarrow ERAIRS;$
- $UPDATE = update \rightarrow UPDIRS;$
- $LOOKUP = delup \rightarrow unlinkup \rightarrow search \rightarrow (DEPLOY[*]LOOKIRS);$
- $DEPLOY = link \rightarrow register \rightarrow DEPIRS;$

- $ERAIRS = updera \rightarrow IRS;$
- $UPDIRS = updup \rightarrow IRS;$
- $LOOKIRS = updlook \rightarrow IRS;$
- $DEPIRS = upddep \rightarrow IRS;$

B. Verification results for the evolution mechanism

We use process EM to express the behavior of the whole procedures in the evolution mechanism. Because the process IRS can be regard as the first process in the evolution mechanism, the process IRS is equal to the process EM.

Based on the PAT, we can verify that the process EM is deadlock-free, divergence-free and nonterminating. The results for the processes of the evolution mechanism are illustrated in Fig. 6.

Assertions	
1	EM deadlockfree
2	EM divergencefree
3	EM nonterminating

Fig. 6. The results for the processes of the evolution mechanism

VII. CONCLUSION

This paper provides a novel evolution mechanism between the dynamic physical applications and the corresponding physical information processing systems in the IoT. The evolution mechanism satisfies three properties which are deadlock-free, divergence-free and nonterminating.

ACKNOWLEDGMENT

Partial work of this paper is supported by the International S&T Cooperation Program of China (ISTCP) under Grant No.2013DFA10690.

REFERENCES

- [1] R. Arun, P. Davy and B. Yolande. *Enabling self-learning in dynamic and open IoT environments*. Procedia Computer Science, pp.207-214, 2014.
- [2] A.P. Athreya, B. DeBruhl and P. Tague. *Designing for self-configuration and self-adaptation in the Internet of Things*. IEEE International Conference Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013.
- [3] Y. Cheng, X. Li, Z. Li, et al. *AirCloud: a cloud-based air-quality monitoring system for everyone*. 12th ACM Conference on Embedded Network Sensor Systems (SenSys), pp.251-265, 2014.
- [4] CS Department NUS. *PAT: Process Analysis Toolkit*. Available: <http://www.patroot.com>.
- [5] C.A.R. Hoare. *Communicating Sequential Processes*. New Jersey: Prentice-Hall International, 1985.
- [6] G.Neil and C. Danny. *Internet 0: Interdevice internetworking-end-to-end modulation for embedded networks*. IEEE Circuits and Devices Magazine, pp.48-55, 2006.
- [7] OECD. *Machine-to-machine communications: Connecting billions of devices*. Available: <http://www.oecdilibrary.org>, 2012.
- [8] L. Rao, C. Fan, Y. Wu, et al. *A Self-Adapting Dynamic Service Management Platform for Internet of Things*. LISS 2013, Springer Berlin Heidelberg, pp.783-791, 2015.
- [9] K. Xie, H. Chen and L. Cui. *PMDA: A physical model driven software architecture for Internet of Things*. Computer Research and Development, pp.1185-1197, 2013(In Chinese).
- [10] H. Zhuge. *Semantic linking through spaces for cyber-physical-socio intelligence: A methodology*. Artificial Intelligence, pp.988-1019, 2011.

Architectural Evolution of a Software Product Line: an experience report

Marcelo Schmitt Laser, Elder Macedo Rodrigues, Anderson Domingues, Flávio Oliveira, Avelino F. Zorzo
School of Computer Science (FACIN) - Pontifical Catholic University of Rio Grande do Sul
Porto Alegre – RS – Brazil

Email: {marcelo.laser, anderson.domingues}@acad.pucrs.br
{elder.rodrigues, flavio.oliveira, avelino.zorzo}@pucrs.br

Abstract—This work presents an experience report on the architectural decisions taken in the evolution of a Software Product Line (SPL) of Model-based Testing tools (PLeTs). This SPL was partially designed and developed with the intention of minimizing effort and time-to-market during the development of a family of performance testing tools. With the evolution of our research and the addition of new features to the SPL, we identified limitations in the initial architectural design of PLeTs' components, which led us to redesign its Software Product Line Architecture (SPLA). In this paper, we discuss the main issues that led to changes in our SPLA, as well as present the design decisions that facilitate its evolution in the context of an industrial environment. We will also report our experiences on architecture modifications in the evolution of our SPL with the intention of allowing easier maintenance in a volatile development environment.

I. INTRODUCTION

Over a few decades, more and more software development companies have been using some software engineering strategies, such as reuse-based software engineering, to develop software with less cost, faster delivery and increased quality. Reuse-based software engineering is a strategy in which the development process is focused on the reuse of assets and on a core architecture, reducing the development effort and improving the software quality. In recent years, many techniques have been proposed to support software reuse, such as Component-based development and Software Product Lines (SPL) [11]. Component-based development is centered on developing a software system by integrating components, where each component can be defined as an independent software unit that can be used with other components to create a system module or even a whole software system. In another way, Software Product Lines are focused on developing a family of applications based on a common architecture and a shared set of software assets, where each application is generated, in accordance with the requirements imposed by different customers, from these assets and shares a common architecture [11].

In past years, SPL has emerged as a promising technique to achieve systematic reuse and at the same time to decrease development costs and time-to-market [9]. One of the main SPL development sub-processes is to use the domain requirements and the product line variability model to define the Software Product Line Architecture (SPLA). The SPLA is a common, high level and generic structure that will be used for all the products derived from the SPL.

In order to take advantage of this approach, we have adopted the SPL concept to support the development of our applications [1] [4] [10]. We have found that, although this enabled the reuse of artifacts, thus reducing the time and cost of development, it incurred in a cost related to the evolution of each artifact, as well as that of the SPLA used to manage this evolution.

In this work we report and discuss our experience in implementing and evolving the architecture of a component-based SPL to derive Model-based Testing (MBT) tools. In particular, we describe how we applied software design patterns [6] to map and to instantiate components, these having their variability managed by another component [5]. Finally, we describe two methods of implementing the variable components (features).

This paper is organized as follows. Section II describes the context where PLeTs SPL was designed and developed, as well as briefly presenting its Product Line Architecture (PLA). In Section III we discuss the main PLA limitations identified along the SPL evolution. In Section IV we present and discuss our approach to mitigate these limitations, as well as describe our PLA in accordance with that approach. In Section V we discuss the related work and Section VI presents the lessons learned along the PLA evolution. Finally, the conclusion and the future work are presented in Section VII.

II. CONTEXT

Our research group on Software Testing¹ has been working to design and develop several testing tools. Our research focus is to investigate innovative ways to mitigate the effort of repeatedly creating custom solutions to apply performance, functional and structural testing. After developing several testing tools, either from scratch or using a limited opportunistic reuse, but which had several features in common, we started a collaborative study with the Technology Development Lab of our partner company to investigate the use of SPL concepts to generate these testing tools. As a result of this study we consolidated our SPL called PLeTs [1] [4] [10]. In this SPL, derived products are testing tools that receive behavioural models as input and give either automatic or manual test scripts as output; these models denote specific test cases, thus leveraging testing teams to follow a model-based approach [14], a process that we describe elsewhere [1].

¹www.cepes.pucrs.br

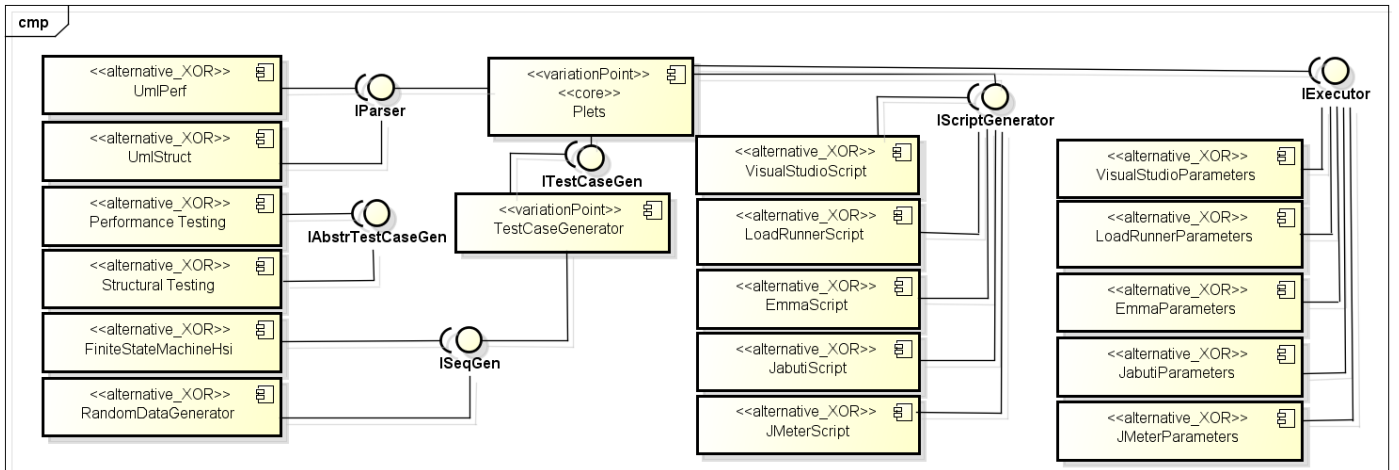


Fig. 1. PLeTs UML Component Diagram

A. PLeTs Architecture

PLeTs was initially designed to support the derivation of a particular testing tool from a set of shared software components, which are then glued together with minimal changes. We defined the use of a replacement mechanism to develop each concrete feature [13] of the PLeTs feature model [1]. In this way, an MBT tool derived from PLeTs is assembled by selecting a set of components and a common software base. We chose this approach to generate PLeTs products because it presents some advantages, such as high-level of modularity and a simple 1:1 feature to code mapping. Figure 1 shows the PLeTs component model.

Since we are using a component replacement mechanism, each provided interface represents a variation point and each variable component implementation represents a variant. The interfaces provided by the PLeTs components are (see Figure 1): a) *IParser*: this interface is a mandatory variation point that has two exclusive variant components, *UmlPerf* and *UmlStruct*; b) *ITestCaseGen*: this interface is a mandatory variation point that has one mandatory component: *TestCaseGenerator*. This component provides two interfaces: *IAbstractTestGen* and *ISeqGen*. The former interface can be realized by one of the following components: *PerformanceTesting* or *StructuralTesting*. The latter interface can be realized by one of the following components: *FiniteStateMachineHsi* or *RandomDataGenerator*; c) *IScriptGenerator*: this interface is an optional variation point that can be realized by one of the following components: *VisualStudioScript*, *LoadRunnerScript*, *EmmaScript*, *JabutiScript* and *JmeterScript*; d) *IExecutor*: is an optional variation point that can be realized by one of the following components: *VisualStudioScript*, *LoadRunnerScript*, *EmmaScript*, *JabutiScript* and *JmeterScript*. For a detailed description of component functionalities, see [3].

In accordance with Figure 1, a valid configuration of a MBT tool derived from PLeTs could have the following components: *PLeTs*, *UmlParser*, *TestCaseGenerator*, *PerformanceTesting*, *FiniteStateMachineHsi*, *LoadRunnerScript*, *LoadRunnerParameters*. Based on the selected components, the generated tool supports the extraction of test information from UML models (*UmlParser* component), then generates test cases (*TestCaseGenerator* component) using a Finite State Machine

and the HSI method (*FiniteStateMachineHsi* component) to apply performance testing (*PerformanceTesting* component). The generated test cases could be used as an input to generate scripts to the LoadRunner testing tool (*LoadRunnerScript* component) and then the scripts could be loaded in the tool and run the test (*LoadRunnerParameters* component).

III. PLETS ARCHITECTURE LIMITATIONS

During the design and development of an SPL, some of the main issues faced by an SPL Architect are caused by changes in the requirements, as these may result in the inclusion of unpredicted features to the SPL (reactive approach [7]), i.e. features that may not fit into the initial design of the SPLA [12]. In our experience, this has often meant that some of the core components of the SPL must be altered to make use of this new feature, which in some cases could imply in changes to the SPLA. This may in turn result in the propagation of these modifications to other unrelated software components in order for them to comply with the changes in the SPLA.

In the early versions of PLeTs [1] [3] [4] [10], we have attempted to solve the problem of SPLA volatility with two different approaches: *Component Interfaces* to map the access between components and, *compile-time definitions* to isolate statements that instantiate variability [2].

For Component Interfaces, we assumed that we would be able to create a coarse-grained relationship between a Feature Model and a Component Diagram, that is, a 1:1 mapping between features and components. Though we were able to create certain components that could be shared between different systems, our final result required high maintenance due to the volatility of our SPLA. Furthermore, any features that could not be directly translated into a single component inevitably became entangled in dependencies, sometimes limiting reuse in the SPL.

To address the difficulties imposed by a coarse-grained relationship, we allowed for the finer-grained representation offered by compile-time definitions. By doing this, we were able to create a more suitable mapping between software features and implemented code. We were also able to concentrate most of the instantiation of variability management within a (comparatively)

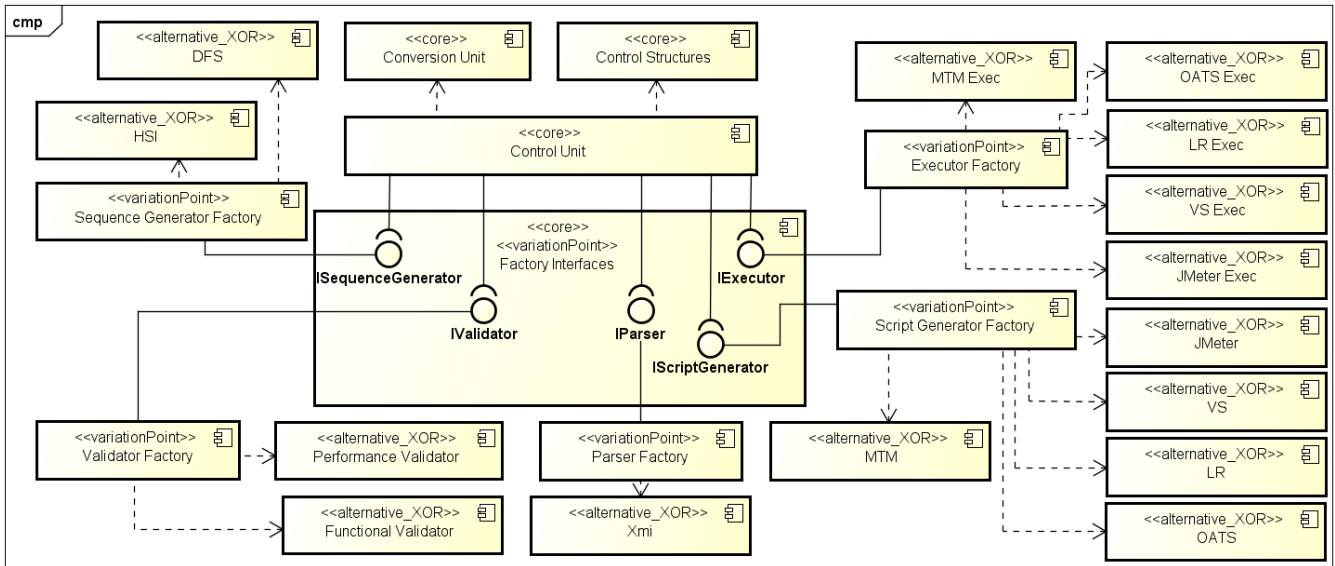


Fig. 2. PLeTs Improved - UML Component Diagram

small section of the code. Before the use of compile-time definitions, we had variability management spread across the entire source code, whereas now we are able to contain it within a well-defined section with few statement blocks.

Another difficulty we constantly faced was the maintenance caused by any changes in a data structure used by more than one component of the SPL. Such changes would often impose the modification of several components. Sometimes these modifications would be of such a degree that the effort required to update all related components was higher than that of creating a new data structure. We soon came to realize that this option was merely mitigating the update effort, as several adapters and converters had to be made over time to suitably enable the new data structure to work with certain product configurations.

Though each of the two approaches described had their merits, neither fully addressed the difficulties we found in evolving the SPLA in our research center environment. This environment is volatile due to the need to assign team members to different projects during certain parts of the development phase and the shifting of team members throughout the development cycle. For an SPL framework to fit into this environment, we found it necessary to establish certain guidelines (see Sections IV-1 and IV-5) and mechanisms (see Sections IV-2, IV-3 and IV-4) for the maintenance and extension of the SPLA.

IV. PLeTs ARCHITECTURE EVOLUTION

In order to tackle the issues raised by our previous development paradigms, we have adopted a mixed approach that is largely based on the use of the Factory Method design pattern [6] to externalize variability points from the implementation of concrete features [13]. We kept the component-based approach. The difference is that our SPL now has a well-defined core that centralizes the variability management, as well as serving as a starting point to the execution of any derived product.

The core of our SPL is composed by four components (see Components marked with the stereotype *core* in Figure

2), described below: *Control Unit*; *Factory Interfaces*; *Control Structures* and; *Conversion Unit*. Our goal has been to simplify variability management as much as possible by ensuring the independence between the SPL core and the feature instantiations, specifying sections (*i.e.* Factory Interfaces component) of the core to manage variability.

1) *Control Unit*: The Control Unit component is responsible for orchestrating the execution of the system, providing access to the functions of those components that implement features and organizing the data structures necessary for the proper execution of the system. It is designed and implemented without any dependencies on components external to the SPL core, which protects it from modifications in them.

2) *Factory Interfaces*: In order to access the components that are external to the core, that is, all features, the Control Unit makes use of the Factory Interfaces component, which is an abstract representation of the variability points of the SPLA. It contains interface definitions for each variability point, each serving as a connection point for a variable component. Each variability point in the SPL Architecture is represented here by one interface.

In Figure 2, we can see that within the Factory Interfaces component we have five interfaces represented. Each of these interfaces, for example *IExecutor* [3], contains a signature to all operations that must be available in a component that fulfills the role of the equivalent abstract feature², for example, the *Executor* feature. All references contained in the Control Unit with the intention of accessing a variable component will be made with one of these interfaces, so that the Control Unit may know what it is able to do without relying on access to the libraries of the variable components.

3) *Control Structures*: To access the data structures held by the components external to the core without establishing dependencies to the libraries that define these data structures,

²In our approach, abstract features represent the variability points of the system.

the Control Unit makes use of the Control Structures component. Like the Factory Interfaces component, Control Structures is an abstract representation of part of the SPLA, in this case, the data structures required by the SPL. While the Factory Interfaces component represents variability, the Control Structures component represents commonality, that is, any representations that are common to two or more data structures of the system. These are abstract data structures that serve as a surface representation of the concrete structures of the SPL. These abstract structures can either be a number of detailed ones that make use of inheritance or a single structure to serve as a placeholder for all others. To enable the identification of a specific instance of a Control Structure by the Control Unit without creating dependencies between it and the data structure components, we have specified that all Control Structures must have an identification attribute. Specifically, we have used an enumerator, created inside the Control Structures library itself, listing the data structures present in the system.

4) *Conversion Unit*: The Conversion Unit is responsible for parsing structures that are equivalent, i.e. any parsing process that does not change the content of a structure, such as the refactoring of a structure to execute different functions or the updating of a structure to a newer version. The Conversion Unit is a subsystem within the SPL, serving as a centralizer to the inclusion and updating of data structures. This part of the SPL core has been essential to the evolution of our project, greatly lowering the effort required to adapt product configurations to changes in data structures and vice-versa. By having all of the converters available to one another, we are able to create a directed graph of possible structural conversions and identify entry points to easily include new structures to the system. Rather than having to create adapters and converters for all combinations of data structures, the Conversion Unit makes use of the commonalities between them.

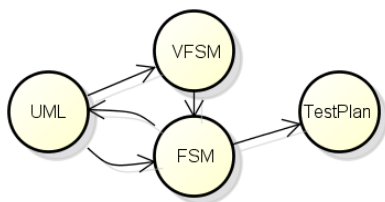


Fig. 3. Conversion Unit Diagram for PLeTs

Figure 3 shows the conversions currently available in the Conversion Unit. We have a single modeling format, based on UML diagrams, and a single test script format, called Test Plan. Additionally, we have the state transition models Finite State Machine (FSM) and (VFSM), from which the test sequences are generated. When the inclusion of VFSM was made, our requirement was that it be convertible to and from the UML and FSM formats, and to the model Test Plan, resulting in a total of five converters. As shown by the cycles in the graph, by implementing two of these converters (“UML to VFSM” and “VFSM to FSM”), we were able to fulfill all five of the requirements for the inclusion of this new structure. For example, the conversion “VFSM to Test Plan” is made by first converting the VFSM into an FSM, and then converting the resulting FSM into a Test Plan. We are aware that this composite conversion of structures may be detrimental to performance, but

in our experience this has not been an issue. Should performance be critical to a certain conversion, a specific converter can be added.

For every abstract structure defined in the Control Structures component, the Conversion Unit has a factory capable of reading its type, as well as the return type desired, and forwarding it to the appropriate concrete structure converter. If a new structure component is developed for the system, specific converters will have to be implemented for that structure in order to convert both to and from it. All of these converters are contained within the Conversion Unit component itself, and are therefore accessible by the SPL Core.

Figure 4 presents an example of the process executed by the Conversion Unit. The Control Unit makes a request to the Conversion Unit. This request sends both the structure to be converted and the desired return type. The Conversion Unit identifies the type of the input structure and forwards the request to the appropriate Converter Factory, in this case the UML Factory. The Converter Factory infers the required converter based on the return type, and forwards the request to it. Finally, the converter casts the resulting structure into a general purpose structure described in the Control Structures component and returns it to the Control Unit.

The key factor that enables us to do this is the extensibility of a single library by partitioning it. The only library upon which the other core components are dependent on is the one containing the factories responsible for the first phase of the conversion, that is, the reading of input type. The concrete converters are each implemented in their own packages, compiled into their own libraries and kept outside of the SPL Core. They are implemented as extensions to the Conversion Unit namespace and included as needed by the compiler, resulting in a single logical unit that is distributed among various libraries with varying dependencies. The result is a final product where the converters, being plugged into the SPL Core as needed, are accessible from all concrete feature implementations while the SPL Core, being guarded from these concrete converters, can function in their absence.

5) *Variable Components*: We have found that the reuse of features for the creation of new product configurations becomes simpler and requires less refactoring effort if the variable components that implement these features are developed autonomously, that is, to be capable of execution as an atomic software unit if given an appropriate input. To make this possible without requiring the SPL core to depend on the libraries of all known variable components, an intermediate entry point is represented in the SPLA in the form of Factory components.

These factories make use of both compile-time and runtime logical operations to return an instance of the correct main feature realization. The compile-time operations are used to define what variable components are available to a given product configuration. The runtime logical operations are used when more than one option is available as a realization of a main feature, evaluating input from the user and the current state of the software to return the correct option. More about the Factory Method design pattern can be found in [6].

Ideally, each new feature added to our SPL as an alternative to represent a variability point in the SPLA will be developed as

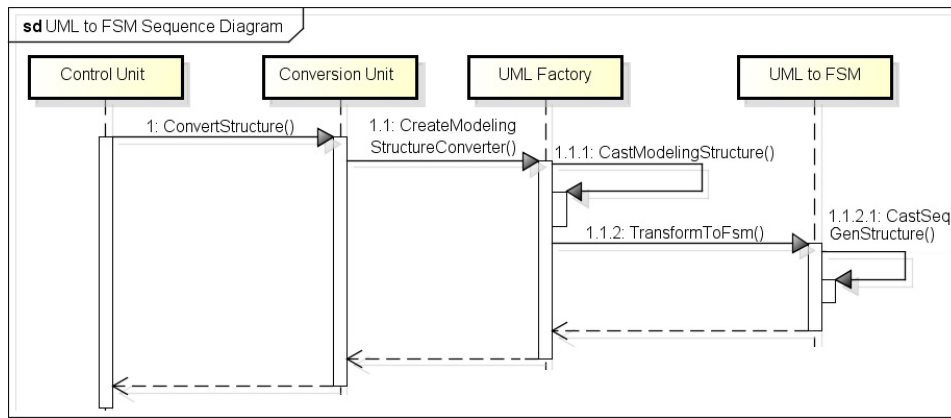


Fig. 4. PLeTs UML to FSM Sequence Diagram

its own component. This results in several diverse components (variants) to resolve each variability point. Alternatively, we have identified the option of representing a variant as an extensible component, in a manner similar to the Conversion Unit (See Subsection IV-4). Based on our experience, this can be useful during particularly hectic periods of development, during which hotfixes are required constantly and there is not enough time to build an entirely new component.

Our experience with the Sequence Generation feature of PLeTs (“Sequence Generator Factory” in Figure 2) serves as a practical example of the application of this option. This particular feature has proven to require adjustments each time a new product configuration was to be derived. Given the speed at which new versions were required and the sometimes mutually-exclusive nature of these adjustments (what would work with one configuration could not work with another, and vice-versa), we found the creation of an entire new component too costly. Instead, we developed small extensions to the existing component that would either include new operations or override existing ones. The original component, along with the extension component, would then be packaged as a single logical unit within a certain configuration, essentially becoming one possible implementation of a certain feature. We used this method purely for reasons of compatibility, and they were used only until such a time as we were able to develop a more robust and versatile component to represent that feature (less demand, more resources available).

In the case of developing new components for each feature, we have the advantage of finer granularity. Since we began using this approach, we have found that the maintenance of our SPL has become simpler than the alternative identified, as the independence of components is strictly enforced by their isolation. This means that the failure or replacement of a concrete feature has little impact to the rest of the system. On the other hand, the stricter isolation requires that the Factory component (responsible for managing the variability point) be kept up to date with modifications to the SPLA, requiring recompilation of the SPL Core. We are aware that this means either that a new Factory component must be written every time a new feature is added to this variability point or that the original Factory’s source code must be available to anyone working on the SPL. In our experience, due to the entire SPL being worked on within a single environment and therefore all

code being available during development, this has not been an issue.

In the case of extensible components we have the advantage of faster development time and greater inter-feature accessibility. Dependencies can be formed between different alternatives to a single variability point, allowing for an incremental extension of libraries without access or change to their source code. This approach also allows a developer to make alterations to a component without necessarily having access to the original factory (indeed, the factory related to this variability point should ideally remain unchanged). This means that independent developers are able to alter and extend the SPL without being given full access to its source code. This approach does not, however, offer any assurances in regards to system granularity. A component built with several codependent sub-components using this approach incurs on the liability to complete system failure if a single unit of the variable component should fail.

V. RELATED WORK

Although Software Product Lines is an active research field, few works present new approaches for the implementation of an SPL, as well as discuss difficulties and limitations faced by SPL design and development teams.

For example, in [15], the authors speak extensively on techniques for Variability Management and present the case study of the Mercure PL, in which the Abstract Factory design pattern is used as a decision model, with each of its concrete factories being related to one product. Our approach has similarities with the one presented in this particular work, but diverges from it in that our use of the Factory Method design pattern is extended to deal with each of the variability points of the SPL.

In [8], this topic is also discussed. A two-dimensional model is proposed for the representation of the issues in variation management, with “files”, “components” and “products” in one axis and “sequential time”, “parallel time” and “domain space” in the other. The author argues that the nine smaller issues defined by this model can be tackled using a divide-and-conquer strategy.

VI. LESSONS LEARNED

This section presents the lessons learned from the development and evolution of PLeTs and the subsequent evolution of its SPLA.

- One of the limitations of PLeTs' previous SPLA was the presence of strong dependencies between components. This often meant that changes in one component resulted in a chain maintenance through all components dependent upon it. To mitigate this problem we have proposed a change to the SPLA to add an SPL core that contains all the basic operations supported by the SPL. Rather than simply making a separation between the concepts of commonality and variability, we found it advantageous to add components into the core for managing the execution and communication of the variable components, *i.e.* the Control and Conversion units. In doing this, we were able to give autonomy to the variable components, avoiding the dependencies that may have arisen between them.
- In some situations, we had difficulty adding new components to the SPL due to the absence of a well-established entry point. In using the Factory Method to automate the process of instantiating components during execution, we were able to isolate the code referent to the majority of variability decisions into small sections that are easy to maintain. This has facilitated the expansion of the SPL by creating an entry point for the adding of new variable components.
- Both SPLAs used benefited from the component-based approach to the realization of variation points in the SPLA. Connecting a new component to the SPL through one of the pre-existing factory interfaces is a simple process, requiring only the packaging of input and output in accordance to the Control Structures component of the core. The new SPLA did not negatively impact in this.
- Should changes in requirements result in new variability points in the SPLA, the SPL core will require modifications. A new factory interface and variation point factory will be necessary, and this will incur in modifications to the Control Unit. We are aware of the implications of such modifications, but have not had the opportunity to test whether or not modifications in the Control Unit would in turn incur in changes to the variable components. Given the well-defined nature of the MBT technique, such radical changes to the SPLA are unlikely in our environment.

VII. CONCLUSION

In this paper we report our experience on the design, development and evolution of a Software Product Line of Model-based Testing tools - PLeTs. We have focused on techniques to simplify the process of managing the evolution of components by use of a software design pattern. We have also presented a description of the core components of PLeTs, which were designed to simplify the communication among variable components, as well as increase the degree of autonomy they have between one another.

Despite the completion of the development of our SPL in accordance to the current SPLA, we are certain that there are further issues to be resolved related to the growth of the SPL. This work details the maintenance and evolution given

to address specific issues identified during the evolution of the SPLA, but over the course of this maintenance the SPL did not have any important features added to it. It is important to evaluate how well the SPLA design proposed here withstands the problems imposed by the addition of new variability points. It would also be valuable to investigate how well a new team member would adapt to this new approach, considering that we work in a volatile environment and new developers might not have prior knowledge of the design pattern used to develop PLeTs.

ACKNOWLEDGMENTS

Study partially developed by the Research Group of the PDTI 001/2012, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91. We thank Dell for the support in the development of this work.

REFERENCES

- [1] L. T. Costa, R. Czekster, F. M. Oliveira, E. M. Rodrigues, M. B. Silveira, and A. F. Zorzo. Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models. In *24th International Conference on Software Eng. and Knowledge Eng.*, pages 112–117, 2012.
- [2] K. Czarnecki and U. W. Eisenacker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
- [3] E. de M. Rodrigues, L. Passos, F. Teixeira, A. F. Zorzo, and R. Saad. On the Requirements and Design Decisions of an In-House Component-based SPL Automated Environment. In *26th International Conference on Software Eng. and Knowledge Eng.*, pages 483–488, 2014.
- [4] E. de M. Rodrigues, L. D. Viccari, A. F. Zorzo, and I. M. Gimenes. PLeTs-Test Automation using Software Product Lines and Model Based Testing. In *22nd International Conference on Software Eng. and Knowledge Eng.*, pages 483–488, 2010.
- [5] C. Gacek and M. Anastasopoulos. Implementing product line variabilities. In *Symposium on Software Reusability: Putting Software Reuse in Context*, pages 109–117, New York, NY, USA, 2001. ACM.
- [6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [7] C. W. Krueger. Easing the transition to software mass customization. In *4th International Workshop on Software Product-Family Eng.*, pages 282–293, 2002.
- [8] C. W. Krueger. Variation management for software production lines. In *Software Product Lines*, pages 37–48. Springer, 2002.
- [9] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., 2005.
- [10] M. B. Silveira, E. M. Rodrigues, A. F. Zorzo, H. Vieira, and F. Oliveira. Model-Based Automatic Generation of Performance Test Scripts. In *23rd International Conference on Software Eng. and Knowledge Eng.*, pages 1–6, Miami, FL, USA, 2011. Knowledge Systems Institute Graduate School.
- [11] I. Sommerville. *Software Engineering*. Pearson/Addison-Wesley, 2011.
- [12] M. Staples and D. Hill. Experiences Adopting Software Product Line Development without a Product Line Architecture. In *Asia-Pacific Software Eng. Conference*, pages 176–183, 2004.
- [13] T. Thum, C. Kastner, S. Erdweg, and N. Siegmund. Abstract features in feature modeling. In *15th International Software Product Line Conference*, pages 191–200, 2011.
- [14] M. Utting and B. Legard. *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2006.
- [15] T. Ziadi, J.-M. Jézéquel, F. Fondement, et al. Product line derivation with uml. In *Software Variability Management Workshop, Univ. of Groningen Departement of Mathematics and Computing Science*, 2003.

Quality Evaluation of Artifacts in Tailored Software Process Lines

Camila Hübner Brondani, Gelson Bertuol, Lisandra Manzoni Fontoura

Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria – UFSM
Santa Maria, Brazil

chbrondani@inf.ufsm.br; gelson.bertuol@gmail.com; lisandra@inf.ufsm.br

Abstract – In software engineering, it is necessary to consider variables such as quality, effort, productivity, time and cost of development. Those variables are negatively affected when defective artifacts are produced. In this case, the cost of rework to correct defects increases in relation to the time of their discovery. Therefore, initiatives should be undertaken in order to find these defects and correct them as soon as they are introduced. This work proposes a mechanism to evaluate the quality goals of software artifacts by means of a quality framework. The study has the objective of organizing concepts that involve the definition of quality goals and their respective methods and metrics of evaluation and can be used to facilitate the task of defining quality plans. Besides that, the framework includes a process to evaluate software artifacts generated from a Software Process Line (SPrL). A Web tool that uses SPrL was used to facilitate the adequacy of the process to different contexts of projects.

Keywords-component: *Software Quality; Software Artifacts; Process Tailoring; Software Process Lines.*

I. INTRODUCTION

The use of software development processes adjusted to the needs of the project and of the work team has strong influence in the final quality of the produced products. This is due mainly by the fact that the process management and the search for continuous quality improvement tend to generate less defective software and within the expected patterns. On other hand, Al-Kilidar *et al.* [1] affirm that, instead of trying to measure the software quality as a whole, should be sought an evaluation of attributes that compose a product which, when combined, may offer a general notion about their quality.

Among diverse approaches described in literature to define and evaluate the quality of software products, stand out the quality models. They search to structure quality in punctual and easy factors to be analyzed and, at the same time, provide a good characterization of such elements [2]. Quality models can be used to evaluate the final product or the different artifacts produced along with the software development. Quality plans may be created from artifacts selected to compose a tailored process. SPrL turns easy the processes tailoring enhancing quality and adequacy of generated processes, decreasing the risks of an inadequate tailoring and even potentializing the reuse.

Software Process Lines (SPrL) imply that organizational process may be organized according with their similarities and variabilities, allowing the composition of processes based in projects specific needs [3]. In this work, the reuse of elements applied to create a quality plan and the composition of processes are supported in a process line based in the context of the project.

In this sense, this work presents an approach to the evaluation of artifacts generated and/or transformed by several activities that compose a tailored SPrL. The approach has the finality to detect and correct possible problems or defects found previously their spread, reducing rework and improving the quality of final products. The proposal is based in a quality framework, structured from a metamodel that relates the evaluation process to the characteristics that involve the artifacts, such as their purposes, interests, methods and metrics. This work also uses lessons learned to form a repository able to help future evaluations of quality artifacts.

This article is organized as follow: in Section II it is presented important concepts to the understanding of this work. In Section III it is described an overview including: a) a metamodel to evaluate artifacts in tailored software processes, b) an approach to generate tailored software process lines and c) evaluation process of artifacts using a tailored software process line. In Section IV, a case study to illustrate the use of our approach is presented. In Section V presents final remarks and some future steps of this work are also discussed.

II. BACKGROUND

Approaches based on knowledge reuse are widely used on software process tailoring, with the purpose of decreasing delivery deadlines and costs, aside from improving the final product quality [4]. The adoption of SPrL allows us to leverage the reuse of individual process components for full process architectures, comprising several inter-related components [3][5][6]. Afterward that, the objective of elaborating consistent processes, once several tailoring approaches are limited to selecting the elements for the process according to the characteristics or the product, and there is no worries with the sequencing and consistency of the generated process. Authors like Jaufman and Munch [7], Washizaki [5] and Barreto [6] propose the use of SPrL as a way to make possible the reuse of software process components.

The authors would like to thank Fapergs (Fundação de Amparo à Pesquisa do Rio Grande do Sul) for the financial support to this work.

At the same time, it is believed that to achieve a final product adequate to the project requirements, the quality should be evaluated along the whole software development process. Quality models are, in general, classified as Hierarchical Models, Conceptual Models and Context Models. The Hierarchical Quality Models describe the relation among a fixed group of high level quality factors, product attributes and appropriate metrics to achieve these factors [8]. They are organized in pillars which are decomposed and refined in specific quality attributes, capable of being evaluated quantitatively by appropriate metrics. The most relevant hierarchical models to this research which, at the same time, where useful as theoretical foundation to the development of others more complex and elaborated models aside from standards and international quality patterns are: the ISO/IEC 9126 [9], the McCall's quality model [10] and Boehm's [11].

The Quality Conceptual Models are used not only in the process evaluation, but as for the development environment as a whole [12]. However, this approach allows not only the evaluation of artifacts, but also the development of team knowledge, the modeled domain, the modeling languages, as for many others aspects that involve the software process building. Two of the most important proposals around the Conceptual Models are the Lindland *et al.* [13] and Krogstie *et al.* [14]. Although the conceptual models may have a high level of abstraction and, therefore, it is more difficult of being practically applied, they can become a good alternative to formalize quality intentions. Other studies have also been proposed to improve the software products and processes qualities during time. These works approaches the evaluation of quality in scientific contexts, normally by means of a framework instantiated from a quality metamodel.

III. PROPOSED APPROACH OVERVIEW

This work presents an approach of quality evaluation of software artifacts generated from a tailored SPRL. To achieve this goal, in this section are described: a metamodel with the objective of organizing quality concepts, a systematic process tailoring using SPRL and an evaluation process.

A. Metamodel of Evaluation of Artifacts in Tailored Software Processes

The metamodel (Figure 1) has the objective of helping who is interested in an adoption of a common vision about the requirements of quality intended for a specific project, at the same time that allows a structured decomposition of elements, concepts and relationships necessary to this vision. The metamodel definition was based in three basic requirements, proposed by Trendowicz e Punter [8], which are: flexibility, reusability and transparency.

The created metamodel, called Quality Metamodel for Tailoring Process (QMTP), represents a group of elements considered pertinent for the quality evaluation of software artifacts. The structure, illustrated at Figure 1, as well as its relations, was made from the quality models described in the Section II, and aims on organizing the evaluators' knowledge in the search for evaluation metrics and methods which better represent its quality purposes.

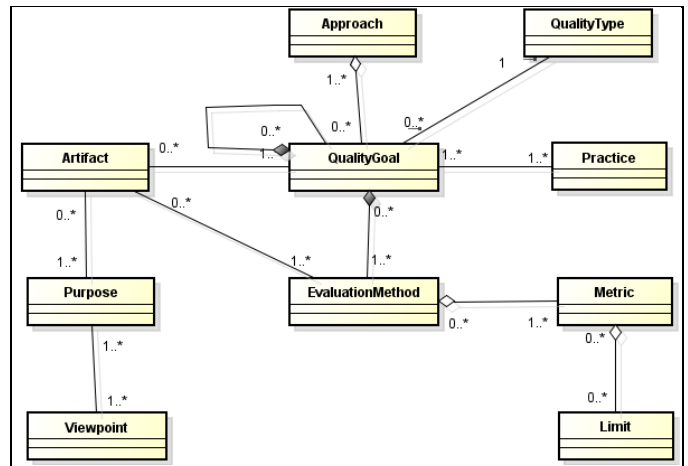


Figure 1. Quality Metamodel for Tailoring Process (QMTP)

The metaclass **Approach** is specified to represent instances of different methodologies or paradigms that can be used in the software development. **QualityGoal** is the definition clear and comprehensible of which attributes or quality characteristics of a certain stakeholder is interested for a certain software artifact. These goals are specifics for each **Artifact**. For example, a Use Case Model has as main objective the comprehension of software requirements, while a source code must be complete and consistent. Therefore, in the context of this work, an artifact is basically associated to the delivers which occur during a software process and the propose of formalizing these elements aims, first, at determining which of them can identify the expected quality needs for the product based on the stakeholders expectations.

Thus, to facilitate the evaluation process tailoring and organize the artifacts based on organizational standards, the metamodel allows that the quality goals can be identified by a **QualityType**. That helps to architecture it on different abstraction levels, which can be created by the evaluation team based on some of the existing models. The **ViewPoint** metaclass is used to identify the stakeholders interested in intended quality goals.

At the same time, the **Purpose** metaclass has the objective of identifying the purpose that describes the artifacts' intention inside software process life cycle, together with the reason why it should be evaluated. The **EvaluationMethod** can be quantitative or qualitative and it identifies how a certain quality goal may be evaluated. These methods are widespread to support specific methods (for example, simulations, inspections, checklists) as well as specific **Metric**, for example, NUC (number of classes per use case) or NCU (number of use cases per class) for UML models or KLOC (number of errors by a thousand code lines). For this, each metric is defined based on a unit and in the minimum and maximum limit values and a value known as acceptable (**Limit** metaclass). Each metric has its own particularities. The definition of values and unities, although recommended, it is optional and that the values attributed in the insertion of a certain metric will serve just as reference to the evaluators, which can change it at the moment in which they are defining the quality plan.

B. Support Approach in the Software Tailoring Processes

This project has begun with the work of Lorenz [15], which objective was to define a systematic approach to the software process tailoring from SPrL and information about the projects' characteristics. The approach allows the reuse of process elements, previously defined, enabling the definition of agile and planned processes. In order to validate the work, it was developed a Web tool called Metamodel for Tailoring Process tool (MfTPt) for supporting the tailoring process, improving the element selection technique for reusable processes. This tool has two modules. The main module has all the functionalities for the creation of process elements repository, as: registration of artifacts, tasks, roles, activities, tailoring requirements, attributes for contextualization of project activities and definition of process architectures. The tailoring software process module is systematized by four steps.

1) *Definition of project characteristics:* the criteria for projects contextualization were described from Octopus Model, proposed by Kruchten [16]. In this model, a software project is characterized by the following attributes: size, stable architecture, business model, team distribution, rate of change, age of system, criticality and control.

2) *Selection of the tailoring requirements and of the process architecture:* this step aims to select process elements that will meet those needs and incorporate them into the process. For each component defined in the architecture, activities that have similar purposes are recovered. An architecture from agile or planned approaches can be defined. It is also possible to combine the features of agile and planned methods in a unique process of development to create a hybrid process.

3) *Prioritization of the activities:* the components defined previously for the architecture are retrieved according to the tailoring requirements for creating SPrL. Thus, the components are prioritized by an algorithm based on the technique Analytic Hierarchy Process (AHP).

4) *Creation of the tailored software processes line:* from the defined process architecture and recovered activities, prioritized and selected, there has the development of tailored SPrL according to the situational characteristics of the project. The Figure 2 shows the SPrL for the discipline of requirements.

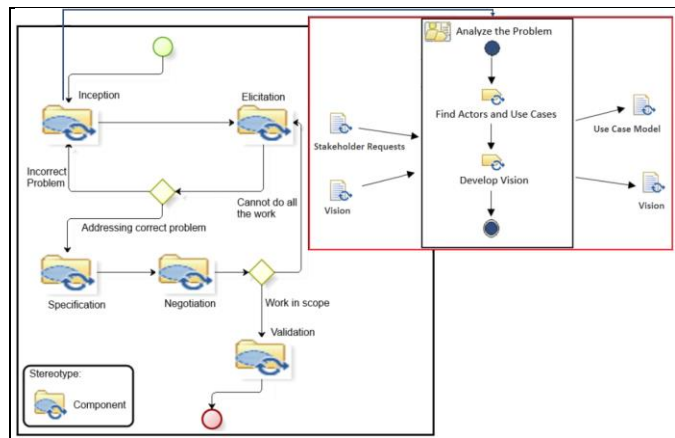


Figure 2. Example of SPrL from requirements discipline

This discipline begins with the component "Inception", goes through the component "Elicitation", "Specification", "Negotiation" and "Validation". For each of these components there is one or more selected activities that contain similar situational context. In the Figure 2, it is highlighted as an example of component the activity "Analyze the Problem". This activity is selected to be instantiated in the component "Inception", with its input and output artifacts.

C. Tailored Software Process Line Artifacts Evaluation Approach

This section describes, as part of the proposed quality framework, an evaluation process for software artifacts of the tailored line. The process has been built upon the standard ISO/IEC 14598 [17]. The objective was to organize the data, structured by the metamodel, so that the evaluators have a solid and practical reference at the time to validate the artifacts produced during the project based on quality goals they think are the most important to the final product.

As a first step to determine the quality plan, there is a need to populate the repository. The data can be extracted from many different sources, including models, specific work or even experts experience or previous projects developed by the organization. You should then define the elements of quality, and for each item there are instances already registered and the possibility of keeping each one of them or inserting new ones. In the Figure 3 there is an example on the use of the module Evaluation of the Quality of the Artifacts of MfTPt according to Figure 1.

Figure 3. Example of the use of the module Evaluation of the Quality of the Artifacts of MfTPt

It was proposed an instantiation of objects for the evaluation of models of Use Cases in order to improve communication of a software project developed under the model-driven paradigm. The goal of main quality is usability, based on internal quality proposed by ISO/IEC 9126 [9]. However, the choices of what purposes, evaluation methods, metrics and practices are best suited to each situation is the prerogative of quality analysts.

The step that corresponds to the evaluation process begins with the definition of the life cycles that make up the used process model. From each phase of the project the assessment requirements are established for the definition of the quality evaluation plan. That is, for each phase of the process life cycle, the generated artifacts are selected, and for each artifact, the evaluators define its purpose, stakeholders and related quality goals. Then there is the review of the specification that involves relate evaluation methods, metrics and practices for each selected artifact.

The last step comprises the documentation of procedures defined and that will be used by the evaluators to define the quality of the selected artifacts, that is, at this stage it produced the quality plan. The idea is that all artifacts settings and their relationships are structured in a clear and understandable way in order to guide the evaluators in quality validation. Finally, the quality plan will be stored and can serve as reference for future assessments, as the artifacts generated during a software process are similar to keeping the same development approaches and the same models of software process.

IV. CASE STUDY

In order to validate the proposal put forward in this paper, we performed a case study involving the evaluation of quality in a software project, developed by undergraduate students of Information Systems at the Universidade Federal de Santa Maria (UFSM). Though it had academic purposes, the project was implemented and deployed in a client. An incremental development process was adopted, consisting of development cycles, where each cycle in the following activities were carried out: definition and requirements analysis, design, coding and testing. The following artifacts were generated: Project Charter; UML models – Use Cases, classes, components diagrams and data model, in addition to source codes.

The project aims to develop a software to manage the payment of mentoring grants of an education institution called Instituto Federal Farroupilha (IFF). This institution has eight geographically distributed campuses. The application's goal is to computerize the management system and the payment of fellow teachers who work on campus.

The case study took place in two stages. The first involved the undergraduates who developed the project and the second had the support of students of graduate in Computer Science from the same university. Training was offered addressing concepts of quality, existing quality models, importance of assessing the quality of software products (and their artifacts as well) and the objective of this work was explained. Then, in possession of a handout provided, the project team aimed to point out, in the group opinion, what were the most important quality goals for the project as a whole and for each of the artifacts created during the project, and stakeholders roles and what means known to evaluate the defined goals.

In response, there was some compatibility between the quality goals chosen in the first phase with those presented in the second phase of the experiment. However, respondents indicated having a high degree of difficulty to suggest possible methods of assessment for each quality goal. The groups were asked regarding the importance of evaluating the quality of software products and as how much a specialized tool could help development teams in the definition and implementation of quality in these products. In response, respondents converged their ideas arguing that the assessment of quality can lead to reduced incidence of errors, reducing costs and production time and positively impact the developer organization's image. At the same time, there was consensus that an aid tool in defining and assessing the quality of software products is essential in the application of quality concepts.

V. CONCLUDING REMARKS

This work presented a proposal of artifacts evaluation process that is inserted in a SPrL tailored from the reuse of properly characterized activities. The SPrL tailored uses a process architecture for which activities to compose the SPrL are retrieved and prioritized. The Web MfTPt tool was developed to support the tailoring process, improving the technique of selecting the reusable process elements and helping the process sequencing, optimizing the resources and improving the process management. The quality plan for artifacts evaluation is developed considering the group of artifacts selected during the process tailoring process and from the reuse of instances of the quality metamodel.

The validation of the proposed approach was accomplished by means of a case study involving a software project. Between the future works, it is intended to validate the proposed approach in real projects, in addition to develop an automation of elements that comprise the artifacts evaluation plan, because it is dependent on human intervention, in other words, on the evaluators knowledge and experience.

REFERENCES

- [1] H. Al-Kilidar, K. Cox, and B. Kitchenham, "The use and usefulness of the ISO/IEC 9126 quality standard," *International Symposium on Empirical Software Engineering*, pp. 126–132, 2005.
- [2] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, "Software quality models: purposes, usage scenarios and requirements," *7th International Workshop on Software Quality*, pp. 9–14, 2009.
- [3] D. Rombach, "Integrated software process and product lines," *International Conference on Unifying the Software Process Spectrum*, pp. 83–90, 2005.
- [4] L. M. Northrop, "SEI's software product line tenets," *IEEE Software*, vol. 19, no. 4, pp. 32–40, 2002.
- [5] H. Washizaki, "Building software process line architectures from bottom up," *Product-Focused Software Process Improvement*, vol. 4034, pp. 415–421, 2006.
- [6] A. S. Barreto, L. G. P. Murta, and A. R. C. da Rocha, "Software process definition: a reuse-based approach," *Journal of Universal Computer Science*, vol. 17, no. 13, pp. 1765–1799, 2011.
- [7] O. Jaufman and J. Münch, "Acquisition of a project-specific process," *Product Focused Software Process Improvement*, vol. 3547, pp. 328–342, 2005.
- [8] A. Trendowicz and T. Punter, "Quality modelling for software product lines," *7th Workshop on Quantitative Approach in Object-Oriented Software Engineering*, 2003.
- [9] ISO/IEC 9126, "Software Engineering - Product Quality," 2003.
- [10] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in Software Quality," *Nat'l Tech. Information Serviel*, vol. 1, 2 and 3, 1977.
- [11] B. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merritt, "Characteristics of software quality," *TRW of software technology*, 1978.
- [12] K. Mehmood, S. S. Cherfi, and I. Comyn-Wattiau, "Data quality through conceptual model quality-reconciling researchers and practitioners through a customizable quality model," *14th International Conference on Information Quality*, pp. 61–74, 2009.
- [13] O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *Software, IEEE*, vol. 11, no. 2, pp. 42–49, 1994.
- [14] J. Krogstie, O. I. Lindland, and G. Sindre, "Defining quality aspects for conceptual models," *International Working Conference on Information System Concepts: Towards a Consolidation of Views*, pp. 216–231, 1995.
- [15] W. G. Lorenz, M. B. Brasil, L. M. Fontoura, and G. V. Pereira, "Activity-based software process lines tailoring," *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 9, pp. 1357–1381, 2014.
- [16] P. Kruchten, "Contextualizing agile software development," *Journal of Software: Evolution and Process*, vol. 25, no. 4, pp. 351–361, 2010.
- [17] ISO/IEC 14598, "Information Technology - Software Product Evaluation," 1999.

BPMN* - A Notation for Representation of Variability in Business Process Towards Supporting Business Process Line Modeling

Marcelo F. Terenciani* Débora M. B. Paiva Geraldo Landre Maria Istela Cagnin†

Facom – Federal University of Mato Grosso do Sul (UFMS) – Campo Grande (MS) - Brazil

E-mail: terenciani@outlook.com, {debora, geraldo, istela}@facom.ufms.br

Abstract

This paper proposes an extension for the Business Process Model and Notation (BPMN), named BPMN, that based on the elements of the feature model (FM), commonly used to represent variability, intends to represent variability in Business Processes Line (BPL). This notation is evaluated by means of an empirical study whose main objective is to compare it with other notation named variant-rich BPMN (vrBPMN) regarding the productivity and correctness of the business process model template (BPMT), which is one of the artifacts that compose a BPL. From the results it was possible to observe that the proposed notation allows the elaboration of BPMT with less errors, although modeling time was kept almost the same.*

Keywords– Business process line modeling, variability, notation, BPMN

1. Introduction

Due to the strong competitiveness in the globalized world, it is necessary for organizations to establish a set of improvements that make their businesses evolve every day. However, for those improvements to be proposed, it is necessary to discover and document business processes from the organizations [7]. In this context, the Business Process Management (BPM) acts as an approach that aims to identify, document, model, operate, monitor and improve business process to achieve results that are aligned with the organizational objectives [2].

The activity of business process modeling supports the BPM, once it promotes better knowledge about organizations business in order to stay competitive in the market [8]; as well as it facilitates knowledge management, since it disseminates how the business functions for all the stakeholders. Regardless of its benefits, this activity is not always

held due to the associated time and costs.

Nonetheless, according to Ladeira [6], business process models can be reutilized, which makes it possible to reduced the time and effort and improve quality in the elaboration of this type of artifact since they had been previously validated and improved. From this perspective, the use of software reuse techniques in the context of business processes, as in the case of BPL, has been utilized to enable the efficient reuse of business process models.

The term BPL emerged from adaptations of concepts and experiences from Software Product Line (SPL) [10] to the context of business processes. The BPL aims to manage a set of commonalities, which are the common parts of the business domain; a set of variabilities (composed by variation point¹, and variant²), that can be selected to accommodate the target process; a set of rules, which explicit the task of decision making to do the flexible composition of business process assets [1].

Basically, the modeling of a BPL is composed by a set of business processes from the same domain (BPL instances); a variability model that represents “what”; and “how” the business varies; a BPMT, which represents the variabilities of business processes in one domain; and a mapping between artifacts, utilized for the traceability between them [7, 3].

Two techniques are commonly utilized to model BPL in the literature [1, 3, 11, 13]: FM [4], utilized to represent variabilities in BPL, and BPMN [9], utilized to represent instances from the BPL. For the BPMT representation there are some notations, discussed in Section 2, but there is no consensus about the most appropriate notation.

The main objective of this work is to present an extension to the BPMN notation, called here BPMN*, which adds elements to the BPMN notation based on elements from the FM, with the intent to allow the creation of BPMT with less errors, once there is need to know another new notation be-

¹Places where the variation can happen (vehicle color, for example) [10].

²Possible existent solutions for the variation point (for example, white, black, silver and red) [10].

*Financial support by Capes

†Financial support by Fundect (T.O. n° 115/2014)

sides the BPMN and the FM. This was observed by means of an empirical study presented in this paper.

2. Related work

Gröner et al. [3] propose the adoption of BPMN elements to model the BPMT and the adoption of the model FM to model the variabilities of a BPL. In that study, each variability is represented in the BPMT in a execution flow of the business process. In this case, it is not possible to explicitly distinguish the commonalities and the variabilities from the BPL in the BPMT, since the BPMN doesn't have specific elements for that purpose.

In another work, Schnieders and Puhlmann [13] propose an extension for the BPMN, referred as vrBPMN, whose objective is to explicitly represent the variabilities on business process models. However, since this extension has many stereotypes, its use implies the need for the business domain engineer to know another notation, in addition to the FM and the original BPMN.

Other notations to represent variabilities in business processes were proposed in the literature, such as the C-EPC [12] and the C-YAWL [5], however they are not based on the BPMN and, thus, not discussed in this paper.

3. BPMN*

The BPMN* notation consists of a extension of the BPMN, since a set of stereotypes and tagged values based on the FM were added to the BPMN notation, as well as a new element was added to the BPMN metamodel to represent a variability association, that is, a relationship between a variation point and its respective variants. The BPMN was chosen to be extended in this work because it's the standard notation to represent business processes. The FM was chosen for being commonly used to represent variabilities [3].

The intent of the BPMN* is to facilitate the modeling of variabilities in business processes of a BPL, that is, the elaboration of the BPMT. The main justification for the proposed extension is that business domain engineers don't need to have knowledge about a specific notation to represent variabilities in business processes, as in the case of the vrBPMN. Besides that, it is believed that the learning curve for the BPMN* is smaller when the business domain engineers already know the BPMN and the FM, which are commonly utilized for the modeling of BPL. That is observed in the results from the empirical study in the Section 4.

Table 1 presents the stereotypes, tagged values and the new element added to the BPMN to represent the variability in business processes of BPLs. A study to identify the BPMN elements where variability could happen was conducted based on [3, 13]. From this study, it was observed

that variability can happen in the following BPMN elements: process, sub-process, activity, event, data object, pool and sequence flow. This way, the elements from the BPMN* can be used to represent variability in such elements during the modeling of variability in business processes.

Elements	Description
<<varpoint>>	Stereotype added to BPMN elements, in order to identify variation points, that is, where the variability happens.
<<variant>>	Stereotype added to BPMN elements, in order to identify variants, that is, the possible resolutions of a variation point. A variant is always associated to a variation point, through the element "Variability Association".
<<mandatory>>	Stereotype added to identify variation points and variants that must be resolved. This stereotype can be omitted.
<<optional>>	Stereotype added to identify variation points and variants that have an optional component.
<<or>>	Stereotype added to variability associations to identify the behaviour of the variation point. In this case, the stereotype indicates that one or more variants from the variation point should be selected.
<<xor>>	Stereotype added to variability associations to identify the behaviour of the variation point. In this case, the stereotype indicates that only one of the variants from the variant point must be selected.
feature	Tagged value added to variability elements to identify a correspondence between elements from the BPMT and the features from the FM.
.....▶	A variability association is an element utilized to represent an association between a variation point and its variants.

Table 1: Elements from the BPMN* notation

Figure 1a represents a model of variabilities in FM, composed by a variation point and its variants. The variation point "Payment" is associated with its variants "Credit", "Debit" and "Cash" through the relationship "or", which allows to select at least one of the variants of the variation point during the instantiation of the BPL.

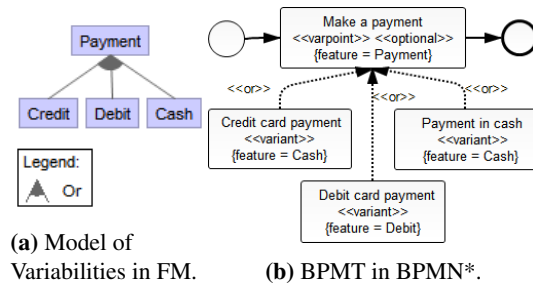


Figure 1: Representation of variation point "Payment".

Figure 1b is the representation of the variation point "Payment" in BPMN*. In this case, the stereotype <<varpoint>> added to the label of the activity "Make a payment" indicates that there is a variation point, the tagged value feature indicates the name of the corresponding feature, enabling the traceability between the FM (Figure 1a) and the BPMT (Figure 1b). The variants are identified with the stereotype <<variant>> and are associated with to their respective variation point through a variability asso-

ciation with the stereotype <<or>> added, indicating the same behaviour from the FM.

4. Empirical Study

The objective of the empirical study is to analyze the BPMTs generated while using the BPMN* and vrBPMN notations, with the purpose of the evaluation in relation to the efficiency in terms of time spent for the elaboration of the BPMT (productivity) and the quantity of errors found in the resulting BPMT (correctness), by the point of view of business domain engineers. The hypotheses are described in Table 2.

Hypothesis	Description
H_0	Time taken to model the BPMT utilizing the BPMN* notation is equal or greater than when using the vrBPMN notation.
H_{a0}	Time taken to model the BPMT utilizing the BPMN* notation is less than when using the vrBPMN notation.
H_1	The quantity of errors made in the modeling of the BPMT utilizing the BPMN* notation is equal or greater than when using the vrBPMN notation.
H_{a1}	The quantity of errors made in the modeling of the BPMT utilizing the BPMN* notation is less than when using the vrBPMN notation.

H: null hypothesis, H_a : alternative hypothesis

Table 2: Empirical study hypothesis

The participants of the empirical study are undergrad students from Computer Engineering, System Analysis, Computer Science, and Technologist in Analysis and Development of Systems courses from the Facom/UFMS.

The participants are divided in two groups balanced by background level and are composed by forty members each. The groups labeled as G-BPMN* and G-vrBPMN, used the notations BPMN* and vrBPMN respectively.

The training was held in two days. In the first day, with the two groups in the same place, an explanation was presented about the basic concepts of the BPMN and FM, with a length of three hours. For a better assimilation of the given concepts, exercises were applied and their solutions provided. In the second day of training, the groups G-BPMN* and G-vrBPMN were separated in distinct places for training about the BPMN* and vrBPMN to be ministered, together with a exercise for the fixation of the terms presented.

With the end of the training, each participant received a table containing the main elements from each notation, according to his or her group, and the correspondence from those elements to the FM; as well as supporting guidelines for the elaboration of the BPMT. They also received three instances of a BPL from the domain of rental services (Process of Video Rental, Process of Borrowing in the Library and Process of Renting Vehicle), the FM of that domain and the execution form. Twenty minutes were given for the participants so that they could understand the provided artifacts and ask any questions related to interpretation.

Even though eighty participants have answered the participant profile form, seventeen of them didn't attend in the day of the study or had given up participating, for not being a mandatory activity in the aforementioned classes. Two participants were removed for not reporting the end time of the elaboration of the BPMT in the execution form; and two were removed because the elaborated BPMT was illegible. Hence, 59 participants were considered during the analysis of the data.

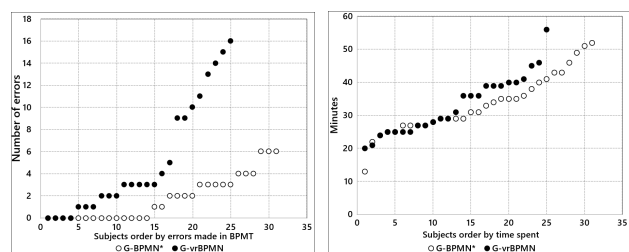
First, an analysis of the data from the study was realized in order to identify outliers. To aid in the identification of outliers, graphs of the type box-plot were generated.

Based on data from the execution form, the time spent by each participant was calculated. This time was used to make a box-plot, where it was detected the absence of outliers related to the time.

A box-plot was constructed taking into consideration the quantity of errors found in the elaborated BPMTs. Hence, the participants that committed 11, 12 and 14 errors in the G-BPMN* group were classified as outliers and were removed from the sample. With the removal of outliers, the group G-BPMN* was left with 31 participants and the group G-vrBPMN with 25 participants, adding up to 56 participants.

Analysis of the data regarding the correctness of the BPMT: All of the BPMTs elaborated by the participants were analyzed with the intent of identifying the errors committed.

In Figure 2a, the participants were sorted in relation to the amount of errors found in the BPMT. It's possible to visualize that members from the group G-BPMN* committed less errors than the members of the group G-vrBPMN. Analyzing the graph it's given that from the eighteen (32,14%) participants that didn't commit errors, 77,78% of them are members of the group G-BPMN*. In relation to the highest number of errors committed, the member from the group G-BPMN* that had more errors, had 10 errors less than the member from the G-vrBPMN that committed more errors. This way, analyzing the two members that committed more errors in each group, it is observed an increase of 166,67% from the group G-vrBPMN in relation to G-BPMN*.



(a) Correctness of the BPMT.

(b) Elaboration time.

Figure 2: Analysis of the data dispersion.

Based on data from the graph presented in the Figure 2a, it's given that the average of errors committed by members of the groups G-BPMN* and G-vrBPMN is of 1.77 and 5.2 erros, respectively. With that, it is observed an average increase in errors of 193,78% for the G-vrBPMN group in relation to the G-BPMN* group.

Analysis of the data in relation to the elaboration time: In the Figure 2b is illustrated a dispersion graph of the time spent by the participants, sorted from the shorter time to the highest time. It's observed that the time spent to elaborate the BPMT with the BPMN* notation tends to be similar to the time spent to elaborate the BPMT with the vrBPMN notation.

Additionally, it is observed that, on average, participants from the group G-BPMN* took 33.35 minutes to elaborate the BPMT, wherein the participant that took less time took 13 minutes and the one that took more time spent 52 minutes in the elaboration. And for the group G-vrBPMN, participants took 33.16 minutes on average to finish the BPMT and, in this group, the participant that finished first took 20 minutes, and the last one took 56 minutes. With this data, it can be observed that the time spent with both notations for modeling the BPMT is approximately the same, even though the group G-vrBPMN is, on average, 0,57% faster than the group G-BPMN*. However, it is noticed an increase in time of 53,8% from G-vrBPMN in relation to the G-BPMN* in regard to the participant that took less time to elaborate the BPMT.

Hypothesis Analysis The hypothesis H_0 was accepted, since the modeling time from the members of the group G-BPMN* was approximately equal (on average, 0,57% slower) to the group G-vrBPMN. Thus the hypothesis H_{a0} was rejected, given that the modeling time of the BPMT making use of the BPMN* was greater than using the vrBPMN. Due to little difference of time spent between both notations, it is noted that other studies should be done to better analyze this hypothesis. The hypothesis H_1 was refuted, since the amount of errors made in the modeling of the BPMT making use of the BPMN* was less (193,78% on average) than when using the vrBPMN. Therefore, the hypothesis H_{a1} was accepted, that affirms that the quantity of errors when using the BPMN* was smaller than when using the vrBPMN. All the artifacts used during the training and execution of the empirical study conducted in this work are available at <http://goo.gl/g4V2TU>.

5. Conclusion and Future Work

This paper presented the BPMN* notation, which is an extension to the BPMN for the explicit representation of variabilities in business process models, useful to support the modeling of BPL. Once that the new elements incorporated into the proposed notation are based on the FM, the

learning curve for its utilization is lower, propitiating the elaboration of BPMTs with a fewer amount of errors, as observed in the empirical study presented. The results from this study also allow to observe that, on average, the modeling time using both notations was about the same for the considered business domain.

As suggestions for future works, there are: *i)* leading of other empirical studies to better analyze the hypothesis H_0 , taking into consideration business processes models from real organizations; *ii)* development of a CASE tool to support the use of the BPMN* notation, aiming to encourage its use; and *iii)* incorporate the proposed CASE tool in an real environment of BPL management with the intent of observe its benefits.

References

- [1] N. Boffoli, D. Caivano, D. Castelluccia, and G. Visaggio. Driving flexibility and consistency of business processes by means of product-line engineering and decision tables. In *3rd Int. Work. on Product Line Approaches in Soft. Eng.*, 2012.
- [2] H. Eriksson and M. Penker. *Business Modeling With UML: Business Patterns at Work*. 2000.
- [3] G. Gröner, M. Bošković, F. Silva Parreiras, and D. Gašević. Modeling and validation of business process families. *Inf. Syst.*, 38(5):709–726, July 2013.
- [4] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA): Feasibility study. Technical report, Software Eng. Inst., 1990.
- [5] M. La Rosa, F. Gottschalk, M. Dumas, and W. van der Aalst. Linking domain models and process models for reference model configuration. In *Business Process Management Work.*, volume 4928, pages 417–430. Springer Berlin Heidelberg, 2008.
- [6] S. Ladeira, R. Penteado, R. Braga, and M. Cagnin. Business modeling reuse based on views: a case study. In *22nd Brazilian Symp. on Soft. Eng.*, October 2008. in port.
- [7] G. Landre, E. Palma, D. Paiva, E. Y. Nakagawa, and M. I. Cagnin. vrBPMN* and Feature Model: An approach to model business process line. In *5th Int. Work. on Process Model Collections: Management and Reuse*, 2014.
- [8] K. Laudon and J. Laudon. *Essentials of Management Information Systems*. Prentice Hall, 2012.
- [9] OMG. Business process model and notation (BPMN).
- [10] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
- [11] C. Rolland and S. Nurcan. Business process lines to deal with the variability. In *43rd Hawaii Int. Conf. on System Sciences*, 2010.
- [12] M. Rosemann and W. M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, Mar. 2007.
- [13] A. Schnieders and F. Puhlmann. Variability mechanisms in e-business process families. In *9th Int. Conf. on Business Information Systems*, 2006.

An Architecture Description Language for Dynamic Service-Oriented Product Lines

Seza Adjoyan
UMR CNRS 5506 / LIRMM
University of Montpellier
Montpellier, FRANCE
adjoyan@lirmm.fr

Abdelhak Seriai
UMR CNRS 5506 / LIRMM
University of Montpellier
Montpellier, FRANCE
seriai@lirmm.fr

Abstract—Reconciling Software Product Lines (SPL) and Service Oriented Architecture (SOA) allows modeling and implementing systems that systematically adapt their behavior in respond to surrounding context changes. Both approaches are complementary with regard to the variability and the dynamicity properties. Architecture Description Language (ADL), on the other hand, is recognized as an important element in the description and analysis of software properties. Different ADLs have been proposed in SOA or in SPL domains. Nevertheless, none of these ADLs allows describing variability and dynamicity features together in the context of service-oriented dynamic product lines. In this sense, our work attempts to describe the changing architecture of Dynamic Service-Oriented Product Lines (DSOPL). We propose an ADL that allows describing three types of information: architecture's structural elements, variability elements and system's configuration. Furthermore, we introduce context elements on which service reconfiguration is based.

Keywords—Architecture Description Language (ADL); Service-Oriented Architecture (SOA); Software Product Lines (SPL); dynamicity; variability; software architecture; Dynamic Service-Oriented Product Lines (DSOPL)

I. INTRODUCTION

Software Product Lines (SPL) and Service Oriented Architecture (SOA) have a common goal from a software development point of view; increase the reusability of existing assets rather than rebuilding new systems from scratch. SPL, on the one hand, allows the development of a family of products that share some common set of core assets [1], [2], [3]. Variability has always been a first concern in SPL studies [16]. According to [4], variability is the ability of a software artifact to quickly change and adapt for a specific context in a preplanned manner. SOA, on the other hand, is a special kind of software architecture, where the main architectural elements are coarse grained and loosely coupled services that are dynamically composable and inter-operable [5]. Being able to modify the architecture of a running system at such a high level of abstraction renders the system highly extensible, customizable and powerful [6].

Variability and dynamicity are core properties to develop complex adaptable software systems such as telecommunication, pervasive, crisis management, surveillance and security systems. In such systems, due to environment changes, a dynamic re-configuration should be carried out without having to re-deploy the whole system.

Combining SOA and SPL constitutes the answer to this need [7]. SOA offers, through its encapsulation property and its explicit interfaces, a solution for achieving dynamic product lines. SPL offers, via variability modeling, analysis and design of changing points in service-oriented architectures.

Architecture Description Language (ADL) is a formalism that allows the specification of system's conceptual architecture [8]. It enables architects to describe and validate systems against stakeholders' requirements from one side, and ease the development and implementation process of complex systems, from another side. It often has a graphical representation or plain text syntax. Conventional ADLs support only static architecture description [6]. Some ADLs provide special formalism for SOA to describe service dynamicity or for SPL to describe variability. Unfortunately no ADL supports the crosscutting SOA and SPL concepts.

To overcome this limitation, we propose an XML-based ADL that allows describing the architecture of a Dynamic Service-Oriented Product Line (DSOPL). It describes the four following elements: (i) the structural elements of a family of software products (i.e. services and connections), (ii) an architectural variability model (i.e. variability points and alternatives), (iii) context information, in addition to (iv) an architectural configuration model (i.e. reconfiguration rules based on context and variability). We choose to use XML as a description language to facilitate understandability and analysis of the described architecture. In addition, XML-based description facilitates tool-support design and interoperability.

The remainder of this paper is organized as follows: In section 2, we discuss related works regarding variability and dynamicity properties. In section 3, we characterize our proposed DSOPL-ADL's elements and demonstrate their utility through a running example. Finally, in section 4, we summarize our contribution and provide directions for future research.

II. RELATED WORK

A. ADLs specifying dynamic properties

A software architecture can be classified in terms of its capability of evolution into two categories: static or dynamic. A static architecture reflects the static structure of software and is completely specified at design time [6], whereas in dynamic architecture, system may evolve after its compilation [1]. In this type of architecture, in addition to specifying the

system in terms of components, connectors and configurations, it should also specify how these components and connectors are evolved or reconfigured at runtime. This evolution of architecture at runtime may happen under several forms: adding/ removing composing elements, reconfiguring architecture (modifying connections), or upgrading existing composing elements (substitution of composing elements).

ADLs are used to describe the properties of static or dynamic software architecture. Among existing ADLs in the literature, only few of them support dynamic reconfiguration such as C2 [9], Darwin [10], π -ADL [11], Rapide [12], ACME/Plastik [13] and Dynamic Wright [14]. In order to describe a specific configuration in [13], the expression “*on {condition} do {operations}*” is used to toggle between different choices at runtime. To replace an instance of component at runtime, *detach* and *attachments* statements are used in *operations* part to respectively unlink and link components. In π -ADL [11], the architecture is considered dynamic since third party services can be discovered and bound to service broker at runtime.

B. ADLs specifying variability properties

Dynamic Software Product Line (DSPL) extends conventional SPL perspective by delaying the binding time of product’s composing elements (i.e. features) to runtime. It produces autonomous and reconfigurable products that are able to reconfigure themselves to select a valid configuration during runtime [15]. Even though there is no concrete agreement of what aspects a dynamic SPL should exactly treat, most approaches agree that the main characteristic of any dynamic SPL framework is the runtime variability, which provides the following common activities at runtime: managing the dynamic selection of variants, autonomous activation/ deactivation of composing elements, substitution of composing elements and dependency and constraint checking of changed elements [22].

Few existing approaches were concerned about representing an architecture that encompasses variability [16]. xADL [17] is an ADL for modeling runtime and design-time architectural elements of software systems. It is defined as a set of XML schemas. xADL 2.0 integrates product lines concepts in the form of three schemas: *versions*, *options*, and *variants* schemas. Concerning the integration of product lines concepts within xADL; this approach suffers from the limitation of expressing constraints (i.e. requires, excludes) between elements of different variation points. Koala [18] defines “*switches*” in order to dynamically bind the selected component. The main limitation in Koala is its static nature; any deployed configuration cannot be changed at runtime and will require application recompilation, thus it is not suitable for dynamic architectures.

Otherwise, approaches that describe variability in product lines architecture are not based on the service-oriented style. Approaches such as [19], [20] describe system’s architecture in terms of services whether in a dynamic or static ADL. Nevertheless, these ADLs are not able to describe variants.

III. DYNAMIC SERVICE ORIENTED PRODUCT LINE ADL

A. Illustrative example

We will use throughout the paper an illustrative example to exemplify concepts related to our proposed approach. This example is about a simplified online sales scenario between four actors; customer, retailer, warehouse and shipment services, as modeled in Fig. 1. The customer accesses retailer’s website, browses the catalog, selects some items and commands an order. The retailer fulfills customer’s order request and inquires the warehouse to prepare all items of the order. Once the order is prepared, the shipping service handles the delivery of items to the customer.

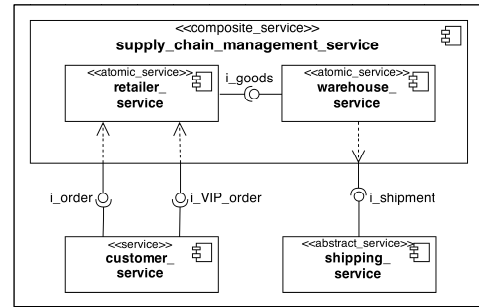


Figure 1. Illustrative example: online sales scenario architecture

B. The DSOP-ADL structure

In order to describe the runtime variability of a Dynamic Service-Oriented Product Line (DSOPL) system at architecture level, we propose an XML-based ADL. This ADL is structured in four sections, as summarized in the schema of Fig. 2:

1. Structural element description: defines all types of the abstract structural entities of the system (services, interfaces, operations).
2. Variability description: here, variation points are defined and also all alternative services of each variation point with the constraints related to each alternative.
3. Context description: variability and configuration descriptions are based on information about context. Thus, information about context elements is described in a specific section of the ADL.
4. Configuration description: here, the rules used to create concrete services and connections are specified to describe how to configure (generate) concrete architectures based on structural, variability and context elements.

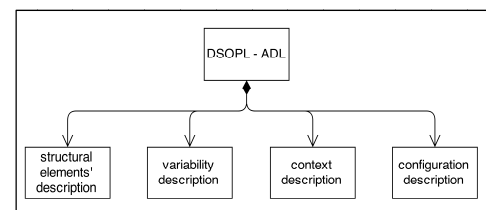


Figure 2. Modular DSOP-ADL

Our approach implicitly separates the four aforementioned architectural concerns from each other. The modular description of architecture in four sections, each of them specifying one type of architectural description has the following advantages:

- 1) It facilitates the modification and re-utilization of each of the four sections of ADL.
- 2) It allows the description and analysis of the architecture by separating the four concerns (structure, variability, context and configuration).
- 3) It allows controlling the traceability links of each type of information among several abstraction levels. For example, the variability described in feature model at requirement level is translated at architecture level through its variability section.

C. Structural elements description

A *service* is an encapsulated and self-contained unit. It interacts with other services through *interfaces*. The system itself is a composite service and is hierarchically decomposed into finer-grained services. Leaf services are called *atomic services*. All other services in the hierarchical tree are considered *composite*. A composite service doesn't execute or implement any functionality by itself, but it delegates this task to one of its composing services. Each composite service is described as sub-architecture.

Each service has a number of provided interfaces and may require a number of required interfaces. *Interfaces* define a collection of methods or operations that are supported by the service. Since services are developed independently from their future exploiting systems, they should have solid and well-defined interfaces that describe their functionalities and operations. Interfaces are two types, either *provided interface* or *required interface*. *Provided interface* of a service is an interface that the service realizes, whereas *required interface* is an interface that the service needs in order to operate. Services communicate to each other through provides/ consumes relationship via their provided/ required interfaces. An interface has a set of *operations*.

The structural description of a service reflects this service meta-model. A service is described based on the following architectural attributes, as shown in Fig. 3: (1) Every service has a name specified by *service_name* attribute. (2) It has a

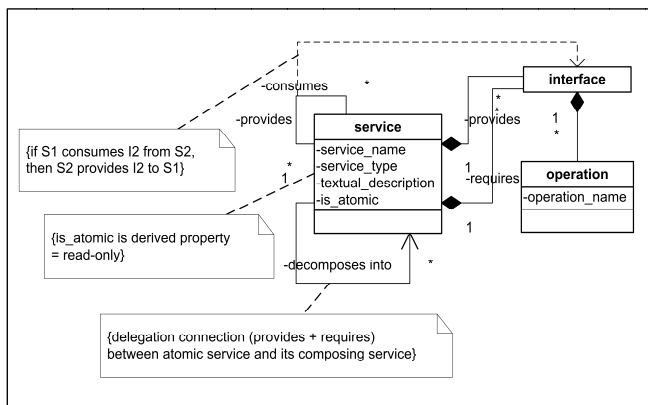


Figure 3. Structural description meta-model of DSOP-ADL

textual_description that explains in plain text the main functionalities of the service, its inputs and expected outputs. (3) *is_atomic* has a Boolean value to indicate whether the service is atomic or composite. Fig. 4 shows the structural section description of the architecture related to our illustrative example.

```
<DSOPL-ADL>
<structural_description>
  <service name="supply_chain_management_service" ... is_atomic="N">
    <interfaces>
      ...
    </interfaces>
    <sub-architecture>
      <service name="retailer_service" ... is_atomic="Y">
        <interfaces>
          <interface name="i_order" role="provides">
            <operations>
              <operation name="submit_order_request" ...> </operation>
              <operation name="get_catalog" ...> </operation>
            </operations>
          </interface>
          <interface name="i_goods_request" role="consumes"> ...
        </interfaces>
      </service>
      <service name="warehouse_service" ... is_atomic="Y">
        <interfaces> ... </interfaces>
      </service>
    </sub-architecture>
  </service>
  <service name="customer_service" ... is_atomic="Y">
    ...
  </service>
  <service name="relay_point_shipping_service" ... is_atomic="Y">
    ...
  </service>
  <service name="home_delivery_shipping_service" ... is_atomic="Y">
    ...
  </service>
</structural_description>
<variability_description> ... </variability_description>
<context_description> ... </context_description>
<configuration_description> ... </configuration_description>
</DSOPL-ADL>
```

Figure 4. Structural description of sales scenario

D. Variability description

Variability in SPL architecture refers to the ability of making changes to system's architecture. We distinct three types of variability:

1) Service variability

It represents binding an alternative service that satisfies pre-conditioned constraints on runtime. Back to our sales example, there are two alternatives of shipment; either a relay point shipment or home delivery shipment, as shown in Fig. 5. The decision of which alternative to choose is taken automatically at runtime depending on customer's selection in addition to other environmental conditions such as the existence of a relay point service in customer's city, as depicted in Fig. 9.

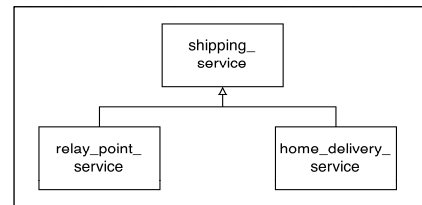


Figure 5. Example of service variability in sales scenario

2) Variability of connection

It may exist several alternative connections between services. The selection of the appropriate connection is done

automatically at runtime according to constraints' satisfaction. For example, the customer service in Fig. 6 can access the retailer service and thus command an order via two different connections; either a connection for a regular customer or a connection for a VIP customer which normally has some extra privileges. The *variation_point* "customer_variation_point" in Fig. 9 is an example of variability of connection.

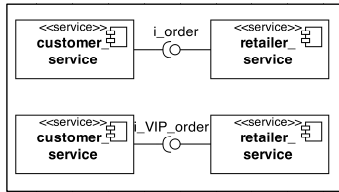


Figure 6. Example of connection variability in sales scenario

3) Variability of composition

This type of variability concerns replacing not only a service or a connection, but replacing a set of interconnected services within a composite architecture. Fig. 7 illustrates another alternative composition of *supply_chain_management_service* than the one in Fig. 1. Here, in addition to the roles of retailer and warehouse services, the manufacturer service realizes requested items and returns them to the warehouse service.

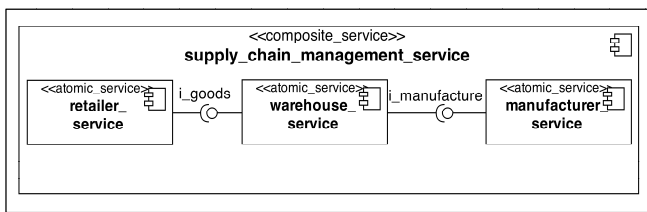


Figure 7. Example of composition variability in sales scenario

The meta-model of variability description is given in Fig. 8. We specify in this section the different variation points that exist in the system at architectural level. A *variation_point* specifies the part of the architecture that can be variable. Each variation point has the following attributes: (1) *variation_name* indicating its unique name, (2) *variation_type* that specifies the type of this variation. Possible values of *variation_type* are either service, connection or composition. (3) *variation_time* specifies whether this variation may occur at compile-time (i.e. before runtime) or at runtime. Contrary to traditional SPL approaches where variability is clearly and completely specified at design time [21], *variation_time* attribute is important in SOA systems, where selection of an alternative during runtime is totally possible. However, some variation points could be specified at compile-time. This reduces the overhead of loading the entire configuration at runtime. Each variation point has several *alternatives*, which are possible elements to fill the selected variation point. Each alternative has a unique name *alternative_name* and an order of priority. This attribute helps the system automatically determine which architectural element is chosen in case there is more than one valid configuration at a given time. The alternative with the highest priority *priority="1"* is the

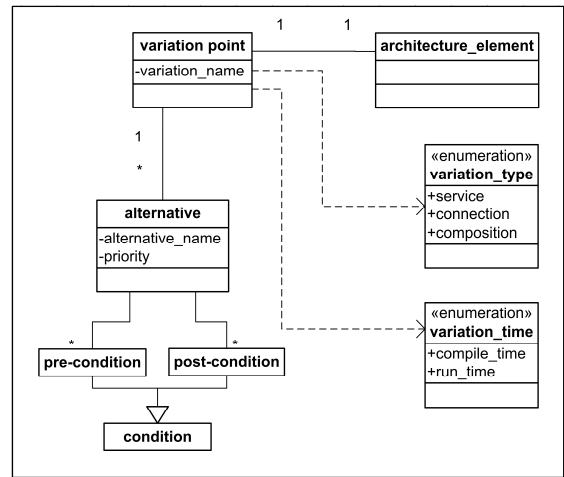


Figure 8. Variability description meta-model of DSOPL-ADL

preferred one in a variation point. Each alternative has a set of constraints, in forms of *pre-conditions* and *post-conditions*, to operate properly. Pre-conditions specify a group of conditions that should be satisfied before executing the selected alternative (i.e. alternative can be selected, only if all constraints of pre-conditions are satisfied). Post-condition represents desirable outcomes when process is completed successfully. Pre and Post-conditions are the equivalent of crosscutting "requires", "excludes" and "and" constraints in Feature Model FM in SPL paradigm. For example, the pre-condition that states that in order to choose the alternative "relay_point_delivery_alternative", the service "relaying_point_service_in_city" should be available (see Fig. 9), this statement is equivalent in FM to a "requires" constraint from "relay_point_delivery" feature to "relaying_point_in_city" feature. On the contrary, condition="unavailable" is equivalent to "excludes" constraint in FM.

```
<variability_description>
<variation_point name="shipping_variation_point"
variation_type="service" variation_time="runtime">
<alternatives>
<alternative name="home_delivery_alternative"
reference_element="home_delivery_shipping_service" priority="1">
<constraints> ... </constraints>
</alternative>
<alternative name="relay_point_delivery_alternative"
reference_element="relay_point_shipping_service" priority="2">
<constraints>
<pre-conditions>
<pre-condition element_type="service"
element="relaying_point_service_in_city" condition="available"/>
</pre-conditions>
<post-conditions>
<post-condition element_type="method" element="re-
calculate_total_amount" condition="execute"/>
</post-conditions>
</constraints>
</alternative>
</alternatives>
</variation_point>

<variation_point name="customer_variation_point"
variation_type="connection" variation_time="runtime">
<alternatives>
<alternative name="regular_customer_alternative"
reference_element="i_customer_order" priority="1"> ... </alternative>
<alternative name="VIP_customer_alternative"
reference_element="i_VIP_customer_order" priority="2"> ...
</alternative>
</alternatives>
</variation_point> ...
</variability_description>
```

Figure 9. Variability description of sales scenario

E. Context description

Architecture reconfiguration is based on context changes. The context consists of any element that influences the behavior and/or the structure of the architecture. It can be related either to system’s environment (e.g. escalator state in the case of crisis management software), evaluated quality of service (e.g. time to response to a query), hardware architecture changes (e.g. server failure), etc. Thus, context element needs to be described in a dynamic ADL. We include these context elements as part of the architecture description to allow context-aware configurations (i.e. autonomous run-time adaptation according to context changes). A context element could capture raw data from a single information source such as a GPS locator that locates customer’s current location to search for a nearby relay point for the shipping service in our sales example. In this case, context element is considered as a *primitive_context*. In some other cases, a single information source could not be sufficient to take decisions; in that case, different atomic information sources’ values are collected, combined and analyzed in order to give sufficient and more accurate information about the context value. We call this context as *composite_context*. We can consider the weather forecast example, where the weather is considered hot when both temperature and humidity sensors exceed a certain threshold.

A simplified meta-model of context is illustrated in Fig. 10. Any context element has a unique `name` and a `context_type` to indicate to which family of contexts it belongs (e.g. contexts related to environment, user preferences, etc.). Context element also has `values_type` that indicates the type of its values, either primitive types such as integer, double, etc. or user-defined types. In Fig. 11, we show two primitive context descriptions from our sales scenario.

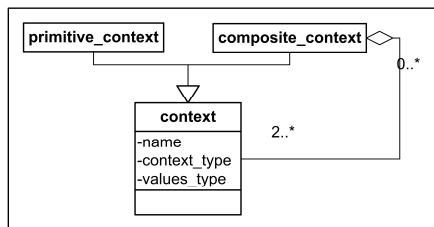


Figure 10. Context description meta-model of DSOPL-ADL

```

<context_description>
<context_type name="environment">
<context_is_aggregate="N">
<name> location </name>
<values_type> double </values_type>
</context>
<context_is_aggregate="N">
<name> shipping </name>
<values_type> enumeration </values_type>
<permitted_values>
<possible_value> home </possible_value>
<possible_value> relay_point </possible_value>
</permitted_values>
</context> ...
</context_type> ...
</context_description>

```

Figure 11. Some context descriptions from sales scenario

F. Configuration description

In traditional architectures, where environment is considered stable, services are selected and composed at design time. In contrast, in dynamic environment, parts of the

software can be instantiated or evolved at runtime. Therefore, we need to maintain, in addition to structural information, architectural information of the running system. The configuration section of DSOPL-ADL allows describing all the configuration rules to generate valid architectures. A valid architecture is a concrete architecture whose services and connections comply with configuration rules.

The configuration description section of DSOPL-ADL has an *initialization* sub-section, where all static elements (services and connections) in addition to alternatives, whose `variation_time="compile_time"`, are instantiated. The `connection` part has two references to two different service interfaces, the one that calls the information `consumer_interface` and the one that provides the information `provider_interface`. The configuration description also has a *dynamic_configuration* sub-section where architectural configurations are triggered based on runtime context conditions. In other words, a concrete architecture is selected through two consecutive execution levels: (1) static bind where core services are selected and bound then (2) late-binding where remaining services and variation points are bound.

In *initialization* sub-section, we first bind static services to the configuration in addition to their connections. In *dynamic_configuration* sub-section, we integrate selected instances of services by observing context changes that are specified in the `condition` part of the configuration rule. Fig. 12 illustrates the architectural configuration meta-model. Any `partial_configuration` has a `name` and an attribute called `priority` of type integer, which determines which configuration to choose in case more than one `partial_configuration` satisfies current conditions. At that time, the one with the higher priority is privileged. Each `partial_configuration` is composed of two parts; `condition` part and `dynamic_action` part. In the `condition` part, we specify conditions that are driven by context elements. In the `dynamic_action` part, we specify all dynamic activities that will be realized. Every action concerns an architectural element which can either be a service or a connection. `Action_type` defines the type of change that will apply on the selected element. Its values are limited to `bind`, `unbind`, `activate` or `deactivate` concerned elements.

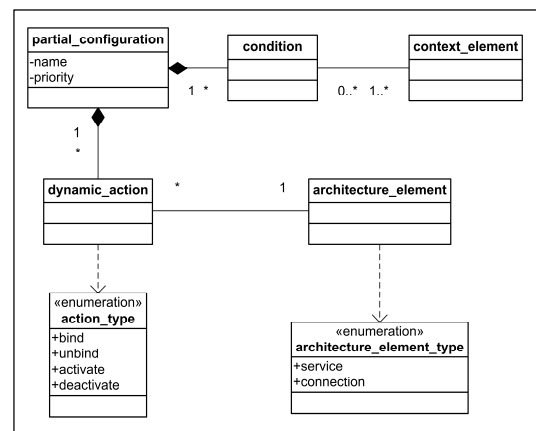


Figure 12. Configuration description meta-model of DSOPL-ADL

In our illustrative example, customer and supply chain management services are instantiated at design time, as depicted in Fig. 13, whereas the relay point shipping service or home delivery shipping service are instantiated dynamically depending on environment's conditions.

```

<configuration_description>
  <initialization>
    <services>
      <deployable_service_instance
service_instance_name="customer_service_instance" ...>
      </deployable_service_instance>
      <deployable_service_instance
service_instance_name="supply_chain_management_service_instance" ...>
      </deployable_service_instance> <!-- when a composite service is
connected, all its composing atomic services are consequently
connected -->
    </services>
    <connections>
      <connection consumer_interface="i_goods_request"
provider_interface="i_goods_response">
      </connection>
      ...
    </connections>
  </initialization>

  <dynamic_configuration>
    ...
    <partial_configuration name="home_delivery_configuration"
priority="2">
      <condition>
        <context_element name="shipping"/>
        <expression operator="equals"> home </expression>
      </condition>
      <dynamic_actions>
        <architecture_element element_type="service"
name="home_delivery_shipping_service_instance" action_type="bind"/>
        <architecture_element element_type="connection"
consumer_interface="i_home_delivery"
provider_interface="i_shipment_ready_delegation"
action_type="activate"/>
      </dynamic_actions>
    </partial_configuration>
    ...
  </dynamic_configuration>
</configuration_description>

```

Figure 13. Configuration description of sales scenario

IV. CONCLUSION AND PERSPECTIVES

We have presented DSOPL-ADL, an architectural language that allows the runtime variability of a service based product lines system to be modeled. To manage the runtime variability of such service based systems at architectural level, we have proposed a modular language called DSOPL-ADL which is structured and composed of four sections; structural, variability, context and configuration. For each part, its meta-model was presented and discussed in detail through an illustrative example.

It is worth noting that we have perceived variability in this work from a spatial perspective and not temporal, that is why we have only considered describing variation points and alternatives and have intentionally eliminated versioning aspect. Another point is that during late binding, we do not use any real-time configuration verification mechanisms. However, we assume that pre-conditions and post-conditions assure a valid configuration.

We are working on generating BPEL process from DSOPL architecture. As a future work; we intend to build a modeling tool for DSOPL-ADL and to conduct more experiments in order to completely evaluate our approach.

REFERENCES

- [1] P. Clements, D. Garlan, F. Bachmann, J. Ivers, J. Stafford, L. Bass, P. Merson. Documenting software architectures: views and beyond, 2nd edition. Addison-Wesley Professional, 2010.
- [2] B. Mohabbati, B. Asadi, D. Gašević, J. Lee. Software Product Line Engineering to Develop Variant-Rich Web Services. In Web Services Foundations, pp. 535-562. Springer New York, 2014.
- [3] P. Clements, L. Northrop. Software product lines: Practices and Patterns. Addison-Wesley, 2001.
- [4] M. Galster, P. Avgeriou, D. Weyns, T. Männistö. Variability in software architecture: current practice and challenges. SIGSOFT Softw. Eng. Notes, vol. 36, no. 5, pp. 30-32, September 2011.
- [5] M. P. Papazoglou, W.-J. Heuvel. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal, vol. 16, no. 3, pp. 389-415, 2007.
- [6] N. Medvidovic. ADLs and dynamic architecture changes. In Joint proceedings of ISAW-2 & Viewpoints '96 on SIGSOFT '96.
- [7] J. Lee, G. Kotonya, D. Robinson. Engineering Service-Based Dynamic Software Product Lines. Computer, vol.45, no.10, pp. 49-55, Oct, 2012.
- [8] N. Medvidovic, R.N. Taylor. A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, vol.26, no.1, pp.70-93, Jan 2000.
- [9] N. Medvidovic, P. Oreizy, J. E. Robbins, R.N. Taylor. Using object-oriented typing to support architectural design in the C2 style. In Proceedings of SIGSOFT '96, pp. 24-32, 1996.
- [10] J. Magee, N. Dulay, S. Eisenbach, J. Kramer. Specifying Distributed Software Architectures. In Proceedings of the 5th European Software Engineering Conference, pp. 137-153, 1995.
- [11] F. Oquendo. π -ADL: An architecture description language based on the higher-order typed π -calculus for specifying dynamic and mobile software architectures. ACM SIGSOFT, pp. 1-14, 2004.
- [12] D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, W. Mann. Specification and Analysis of System Architecture Using Rapide. IEEE Trans. Software Eng., vol. 21, no. 4, pp. 336-355, Apr. 1995.
- [13] A. Joolia, T. Batista, G. Coulson, A.T.A. Gomes. Mapping ADL Specifications to an Efficient and Reconfigurable Runtime Component Platform. WICSA'05, pp.131-140, 2005.
- [14] R. Allen, R. Douence, D. Garlan. Specifying dynamism in software architectures. In Proceedings of the Workshop on Foundations of Component-Based Systems, Zurich, Switzerland, pp. 11-22. 1997.
- [15] C. Cetina, P. Trinidad, V. Pelechano, A. Ruiz-Corts. An Architectural Discussion on DSPL. 2nd International Workshop on Dynamic Software Product Lines (DSPL08). Limerick, Ireland, 2008.
- [16] E.Y. Nakagawa. Reference architectures and variability: current status and future perspectives. In Proceedings of the WICSA/ECSA, 2012.
- [17] E.M. Dashofy, A. van der Hoek, R.N. Taylor. An Infrastructure for the Rapid Development of XML-based Architecture Description Languages. In ICSE2002, Orlando, Florida, 2002.
- [18] R. van Ommering, F. van der Linden, J. Kramer, J. Magee. The Koala Component Model for Consumer Electronics Software. Computer 33, 3, pp. 78-85, March 2000.
- [19] F. Oquendo. π -ADL for WS-Composition: A Service-Oriented Architecture Description Language for the Formal Development of Dynamic Web Service Compositions. In: SBCARS, pp. 52-66, 2008.
- [20] X. Jia, S. Ying, H. Cao, D. Xie. A New Architecture Description Language for Service-Oriented Architecture. In Sixth International Conf. on Grid and Cooperative Computing GCC, pp. 96-103, 2007.
- [21] M. Galster. Describing variability in service-oriented software product lines. In Proceedings of the ECSA'10, pp. 344-350, 2010.
- [22] R. Capilla, et al. An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. Journal of Systems and Software, vol. 91, pp. 3-23, May 2014.

Social Analysis of the SEKE Co-Author Network

Rehab El Kharboutly
Software Engineering
Quinnipiac University
Hamden, CT 06518
Ruby.elkharboutly@quinnipiac.edu

Swapna S. Gokhale
Computer Science & Engg.
Univ. of Connecticut
Storrs, CT 06269
ssg@enr.uconn.edu

Abstract

We extract the co-author network over the entire history of the SEKE conference from 1988 through 2014. In this network, authors represent nodes and a pair of authors is connected by an edge if they have co-authored at least one article over the entire duration. We analyze this network using socio-centric and ego-centric network methods to study the extent to which the authors are involved in the SEKE community, and the patterns of collaboration between them. Socio-centric analysis reveals that most authors publish a very small number of articles, and collaborate within tightly knit circles. In fact, only a tiny fraction of the authors consistently return to SEKE to disseminate their research. Ego-centric measures of centrality confirm these findings by revealing that only a small percentage of the authors are structurally dominant, and influence the flow of communication among others. Based on these findings, we believe that strategically SEKE could benefit from cultivating a wider base of influential authors, promoting broader collaborations, and encouraging one-time authors to return.

Keywords

Co-author network, Clustering Coefficient, Centrality

1 Introduction and Motivation

The International Conference on Software Engineering and Knowledge Engineering (SEKE), now in its 27th year, is a premier conference that aims at bringing together experts in either Software Engineering (SE) or Knowledge Engineering (KE) or both. Specifically, the conference seeks to emphasize the transference of methods between both domains [11]. Since its inception in 1988, SEKE has consistently expanded; both in terms of the number of papers and number of authors by welcoming contributions from traditional SE and KE topics as well as emerging areas.

Participation of researchers and authors is vital to the long-term survival of any conference. A conference is sustained by those authors who consistently return to the venue. However, for healthy growth, a conference should also seek to attract new contributors into its fold, and simultaneously foster collaborations between existing and new authors. Collaborative work offers many advantages, individually for the authors, collectively for the entire scientific community, and finally for the conference itself. It leads to cross-fertilization of ideas, sharing of resources, skills and workload, efficiency in the use of time, and avoidance of competition – all in the pursuit of a mutually shared, common goal. Collaborative research also encourages knowledge sharing, which is essential for knowledge creation, because a person's limited cognitive capability, and bounded rationality [25] imposes a natural limit on what an individual working alone can achieve. Thus, working together can increase the research productivity and impact of an individual [13]. Finally, fostering collaboration can strategically improve the quality, stature, and reputation of a conference, because when a conference spurs the collaboration, it is very likely that the new team chooses the same venue to publish their new joint work.

Co-authorship may be regarded as a strong evidence or an explicit product of collaborative work [10]. In fact, a significant proportion of scientific collaboration leads to co-authored articles. Collectively, such joint authorship of research articles leads to a network, where nodes are authors and links between two authors (nodes) represents at least one joint article between them. Such a co-author network can be considered to be the first-order approximation of complete scientific collaboration network [19]. Therefore, a study of the co-author network can offer insights into whether the social structure of a conference community is conducive to collaborative research. It can also offer suggestions on how such synergistic effort can be promoted.

In this paper, we study the co-author network of the SEKE conference, extracted from the DBLP, KSI and elec-

tronic proceedings spanning years 1988 through 2014. We studied this network using socio-centric and ego-centric analysis methods to understand the degree to which authors are embedded in the SEKE community, and the structural patterns of interactions among them. Socio-centric analysis reveals that most authors have published opportunistically (one or two) papers in SEKE, and only very few return consistently. Moreover, most authors collaborate within their small, tightly knit circles of 2 – 3 collaborators. Approximate power-law spreads of ego-centric measures of centrality, namely, degree, closeness, and betweenness confirm these socio-centric observations by revealing that very few authors are structurally dominant, and control and influence the flow of information and communication among others. Based on these observations, we believe that the SEKE conference could derive long-term benefits by strategically: (i) cultivating a wider base of influential authors structurally embedded in the community; (ii) promoting broader collaborations; and (iii) encouraging one-time authors to return.

The rest of the paper is organized as follows: Section 2 describes data collection and pre-processing. Section 3 and 4 discuss socio-centric and ego-centric analyses respectively. Section 5 surveys related work. Section 6 concludes the paper and offers directions for future work.

2 Data Collection and Pre-Processing

We extract the co-author data from 26 editions of SEKE conferences from its inception in 1988 to its most recent in 2014. Of these years, proceedings for 2013 and 2014 were obtained electronically, data for 1991, 1997, and 1998 from the SEKE website, and the rest from DBLP. Since most data came from DBLP, and was retrieved in XML format, the textual citations obtained from the web and electronic proceedings were parsed and formatted into XML as well. To the best of our knowledge, DBLP does not provide a way to download data to a txt file, so we manually transferred the XML entries to a file. Figure 1 shows an example entry in the DBLP proceedings in the XML format.

We pre-processed this data to replace all the special characters with acceptable XML characters, especially in European names. Authors wrote their names in multiple formats, including first and last name, first and middle initials and last name but the most common representation was first initial, last name. Thus, we translated all the names into this common format. We noticed that many authors shared a last name, but it was very rare (only 4 – 5 instances) for authors to share the combination of first initial and last name. We manually disambiguated between such authors by consulting their affiliations and emails; assuming that authors who share a name but not affiliation and/or email represent different individuals. We added unique tags to identify identical combinations that represent different individuals.

Table 1: Socio-centric Metrics

Metric	Value
Duration	1989 – 2014
Total Number of Authors	2990
Total Number of Papers	1738
Average Papers Per Author	1.49
Average Collaborators Per Author	2.46
Density	0.0059
Average Component Size	4.9
Largest Connected Component	1126
Average Path Length	7.329
Diameter	22
Average Clustering Coefficient	0.852
Number of triangles	4268

After pre-processing, we implemented a parser to extract pairs of collaborators from the XML entries. The pairwise list of authors created by the example XML entry is in Figure 2. This list was checked each time a newly created pair matches an existing entry in the list; if there is a match, the number of contributions for that pair is incremented, otherwise a new pair is added with a collaboration count of 1. We also maintain the number of papers and collaborators for each author. Finally, this list of collaborator pairs is used to create the adjacency matrix. We note that although we keep track of the collaboration count for each pair, for this analysis the adjacency matrix represents an unweighted graph. That is, if authors A and B have co-authored at least one paper, the corresponding element in the matrix is set to 1, otherwise it is set to 0. Altogether we processed 1738 articles written by 2990 authors to build the SEKE co-author network.

B. Cheng, R. Bourdeau
R. Bourdeau, G. Gannod
C. Cheng, G. Gannod

Figure 2: List of Author Pairs – DBLP Entry in Figure 1

3 Socio-centric Analysis

In this section, we discuss socio-centric metrics that are computed over all the nodes in the network. We compare these metrics, shown in Table 1, with those of other scientific communities within and beyond computer science.

3.1 Individual or Local Metrics

Individual metrics are computed locally by considering the immediate connections of each author to understand the

```

<inproceedings key="conf/seke/ChengBG94" mdate="2007-02-23">
<author>Betty H. C. Cheng</author> <author>Robert H. Bourdeau</author>
<author>Gerald C. Gannod</author> <title> The object-oriented
  development of a distributed multimedia environmental information system.
  </title>
<pages>70-77</pages>
<year>1994</year>
<crossref>conf/seke/1994</crossref>
<booktitle>SEKE</booktitle>
<url>db/conf/seke/seke1994.html#ChengBG94</url>
</inproceedings>

```

Figure 1: Example DBLP Entry in XML Format

author’s involvement in the SEKE community. We found that on an average a SEKE author writes 1.49 articles, collaborates with 2.46 others, and on an average a SEKE article has 1.5 authors. These values are low compared to biology (6.4, 3.75 and 18.1) and physics (5.1, 2.53 and 9.7) co-author networks [22]. This difference may arise because biologists and physicists may need to collaborate more frequently and widely due of the experimental nature of their work. However, these metrics are consistent with the values for the co-author network in library and communication sciences (2.40, 1.80, and 2.24) [15], a field that may be closer to SEKE in terms of culture, traditions, practices, and norms. Figure 3, which shows the distribution of the number of authors per article further confirms that a very large percentage of SEKE articles has four or fewer authors. Articles with five or more authors are very rare, with seven being the maximum.

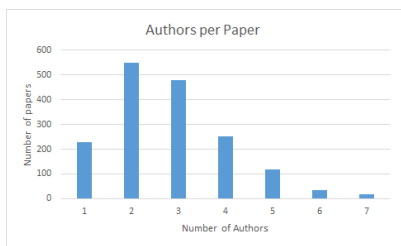


Figure 3: Distribution of Authors per Article

Figure 4, which shows the distribution of number of articles per author indicates that a large percentage of the authors publish only one article, and a very small percentage publishes three or more articles. This approximate power-law spread suggests that while the conference enjoys a very small loyal base, most authors opportunistically choose SEKE. Figure 5, which shows the distribution of the number of collaborators shows that a significant proportion of authors have between 1 and 6 collaborators. A group with two to three collaborators could represent a graduate advisor with his or her doctoral students. A very small per-

centage with no collaborators could represent solo authors. Finally, groups with 5 or more members could represent collaborations across institutions.

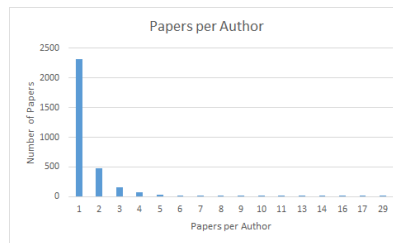


Figure 4: Distribution of Articles Per Author

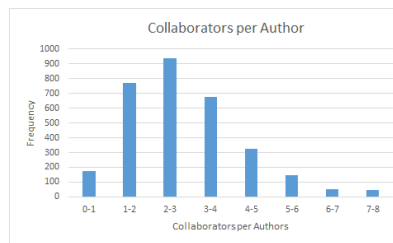


Figure 5: Distribution of Collaborators Per Author

3.2 Aggregate or Global Metrics

Aggregate social metrics, computed by considering the global network, will reveal the degree of closeness of the SEKE community structure.

- **Network Density:** The network density (D) is defined as the number of edges T to the number of possible edges and is given by Equation (1). The density of the SEKE co-author network is 0.0059, indicating an overall very sparsely connected network.

$$D = \frac{T}{N(N - 1)} \tag{1}$$

- **Average Path Length:** This is calculated by finding the shortest path between a pair of nodes and then dividing by the total number of pairs. This shows, on an average, the number of steps it takes to reach one author from the other. The average path length of 7.329 and diameter of 22 indicates that the SEKE network does not exhibit small-world properties, where the average path length and diameter is around 2.0 and 6 respectively. This is surprising because we would expect stronger homophily between SEKE authors, who are mostly computer scientists, than other types of shared interests based on geographic proximity, or organizational affiliation, which typically lead to small-world properties in social networks.

- **Clustering Coefficient:** This measures the degree to which the authors group together so that the probability of a tie between two authors in a cluster is greater than the probability of a tie between any two random authors. The clustering coefficient is defined as the average clustering coefficient of all the nodes [5], where the clustering coefficient C_v for a node v is the proportion of all possible edges between the neighbors of a node that actually exist. The clustering coefficient is based on the number of triangles or closed and open triplets. The average clustering coefficient is a real number between 0 and 1, with the SEKE value being 0.8268. 734 nodes have been excluded from this computation because they have only one edge, and the network has 4268 triangles. The value closer to 1 suggests the presence of a small yet, close community and a large number of isolated groups. The distribution of the clustering coefficient in Figure 6 supports this conjecture, with a large peak at a very high value.

- **Component Sizes:** Similar to other co-author networks, the SEKE network is not a single connected graph. Therefore, to measure the degree of connectivity, we measure the relative size of the largest connected component as its actual size divided by the size of the network, which is approximately 38%. Previously, the relative sizes of the largest connected components were observed to be 20% for the library and communication science [15], 60% for SIGMOD [21], 92.6% for Medline, and 57.2% for NCSTRL [23]. The relative size for SEKE is consistent with Kretschmer’s [12] observation that the largest components usually have a ratio of around 40%. This relative size may also be impacted by the nature of the disciplines, experimental sciences such as biology and physics may have larger connected components compared to computational disciplines such as SEKE and library and communication sciences. The distribution of the connected component sizes shown in Figure 7

has a peak at component size of two, followed by a size of three. This suggests that most authors collaborate within their comfort zone of friends, colleagues or members within their research group, rather than seeking out completely new partners.

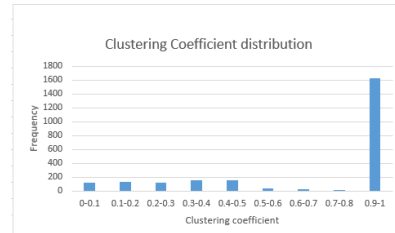
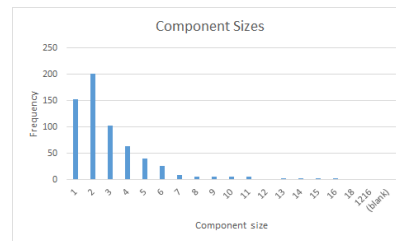


Figure 6: Distribution of Clustering Coefficient



4.2 Closeness Centrality

Closeness centrality is defined as the mean shortest distance by which a given author is separated from all the others [18]. It is measured as the average of the total reciprocal distance of an author from each of the other authors. Closeness centrality of an author v is given by Equation (3), where $d(i, j)$ is the distance between the two authors i and j , and N is the number of authors. A message originating in the most central position (i.e. from the author with the highest closeness centrality) would spread throughout the network in minimum time. Moreover, an author with high closeness centrality could access or obtain the resources owned by others more efficiently than any other author. Therefore, closeness centrality is a surrogate measure of an author's efficiency in communicating with others.

$$C_c(v) = \sum_{k=1}^N \frac{1}{d(i, j)} \quad (3)$$

4.3 Betweenness Centrality

Betweenness centrality is defined as the proportion of the shortest paths between all pairs that pass through a given author [3]. It represents an author's ability to control the flow of resources or information, which enables the author to broker information and resources to others [8]. Betweenness centrality of an author v is given by Equation (4), where $g_{j,v,k}$ is all the geodesics linking authors j and k which pass through author v and $g_{j,k}$ is the geodesic distances between authors j and k . Authors with high betweenness centrality play the role of a "middleman" or a "bridge" and could gain different resources and information from different groups. Also, when authors with high betweenness are removed, it typically results in the largest increase in the distance between others. It thus measures authors' importance to others' virtual communication.

$$C_B(v) = \sum_{j,k \neq v} \frac{g_{j,v,k}}{g_{j,k}} \quad (4)$$

Figures 8, 9, and 10 respectively show the distributions of the degree, closeness, and betweenness centralities for the SEKE co-author network. The spreads of these measures can be approximated using power-law distributions. The distribution of degree centrality in Figure 8 indicates that a large number of authors have a small number of collaborators, and only a fraction collaborate with a large number of others. The co-author network, color coded according to the node degree ranging from 1 to 37, depicted in Figure 11 confirms this distribution. In this network, blue nodes, which make up only a small fraction have the highest degree. Similarly, the distribution of closeness centrality in

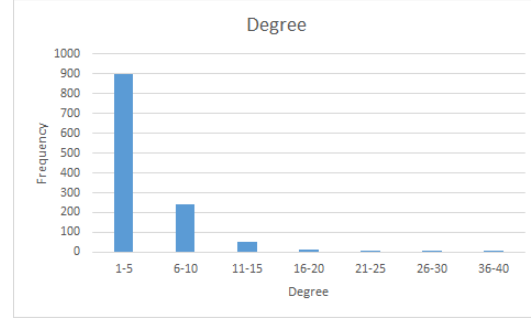


Figure 8: Distribution of Degree Centrality

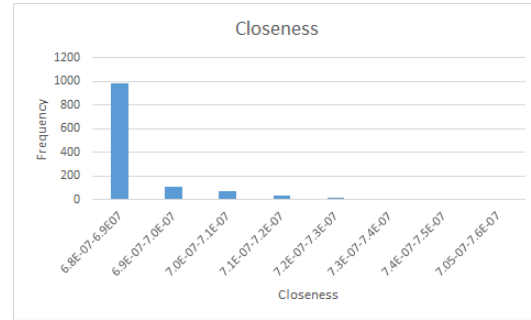


Figure 9: Distribution of Closeness Centrality

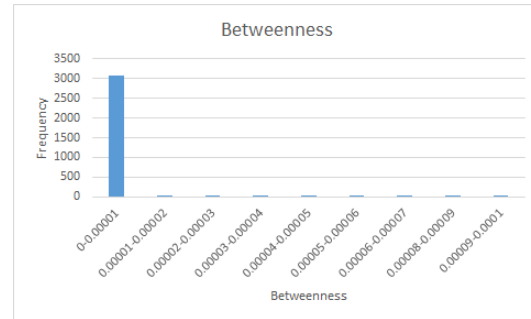


Figure 10: Distribution of Betweenness Centrality

Figure 9 indicates that a very small number of authors are highly efficient in communicating with others and accessing their resources. Betweenness centrality distribution in Figure 10 suggests that after a large spike at the lowest value, the remaining values show the same proportions. Thus, a majority of the authors do not lie on the shortest paths between other pairs. Thus, in summary, although each centrality measures a different aspect of authors' embeddedness, we find that a very small fraction of SEKE authors lie in prominent positions. These authors sport a large number of collaborators, lie on the shortest paths between other pairs, and are highly efficient communicators.

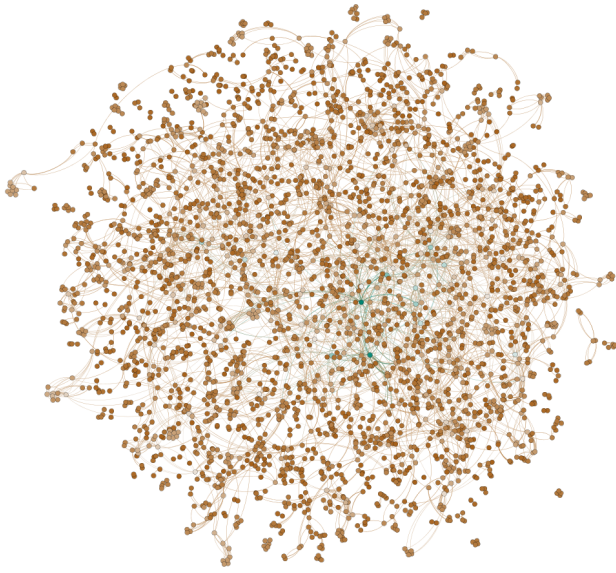


Figure 11: SEKE Co-Author Network

5 Related Work

Social network measures have been used to study the properties of co-author networks in various fields including mathematics, biology, physics, and computer science. Some authors also study how the network structure impacts local or micro-level properties including citation counts. These works, their measures and data, and their key objectives and findings are summarized in Table 2.

Most of the works in Table 2 study readily available, archived data from large communities such as Medline, Physics or Mathematics authors. They also assess the impact of network structure on micro-level properties of individual authors or articles, mostly captured in the form of citation counts. Our work can be distinguished in the following two ways: (i) we extract and process the co-author network of the SEKE conference from three sources; and (ii) we corroborate socio-centric and ego-centric measures to offer recommendations on how the SEKE conference could strategically improve its stature. The SEKE conference, by the virtue of its more than 25 years of history as a premier conference at the interplay of SE and KE, affords this unique opportunity.

6 Conclusions and Future Work

In this paper, we describe the process of extracting the network of SEKE co-authors over the entire history of the conference. We analyze this network using socio-centric

and ego-centric network analysis methods to understand patterns of author involvement and collaboration. Corroborating the results from both these analyses reveals that the SEKE conference is characterized by a large percentage of authors who publish one or two papers, and who collaborate in tightly knit circles. A small fraction of the authors enjoy structural dominance in the network, and control and influence the flow of information and communication. Based on these findings, we offer recommendations that could strategically benefit SEKE.

Our future research involves longitudinal analysis to understand how the patterns of collaboration have evolved since the early editions of the conference.

References

- [1] A. Abbasi, K. S. K. Cheung, and L. Hossain. “Egocentric analysis of co-authorship network structure, position and performance”. *Information Processing and Management*, 48:671–679, 2012.
- [2] J. Bollen, M. A. Rodriguez, and H. Van De Sompel. Journal status. *Scientometrics*, 69(3), 2006.
- [3] S. P. Borgatti. “Centrality and network flow. *Social Network*, 27(1):55–71, 2005.
- [4] M. E. Burkhardt and D. J. Brass. “Changing patterns or patterns of change? The effects of a change in technology on social network structure and power”. *Administrative Science Quarterly*, 35(1):104–127, 1990.
- [5] X. Cheng, C. Dale, and J. Liu. “Statistics and social network of YouTube videos”. In *Proc. of Intl. Workshop on Quality of Service*, pages 229–238, 2008.
- [6] R. P. Dellvalle, L. M. Schilling, M. A. Rodriguez, H. Van de Sompel, and J. Bollen. “Refining dermatology journal impact factors using PageRank”. *Journal of the American Academy of Dermatology*, 57(1):116–119, 2007.
- [7] Y. Ding. “Scientific collaboration and endorsement: Network analysis of coauthorship and citation networks”. *J. Informetr*, 5(1):187–203, 2011.
- [8] L. C. Freeman. “Centrality in social networks: Conceptual clarifications”. *Social Network*, 1(3):215–239, 1979.
- [9] C. N. Gonzalez-Brambila, F. M. Veloso, and D. Krackhardt. “The impact of network embeddedness on research output”. *Research Policy*, 42:1555–1567, 2013.
- [10] B. He, Y. Ding, and E. Yan. “Mining enriched contextual information of scientific collaboration: A meso

Table 2: Research Summary: Co-Authorship Networks

Citation	Measures	Data	Objectives
Mutschke [20]	Centrality	Digital Libraries	Collaboration patterns
Liu <i>et al.</i> [17]	Centrality	Joint Conf. on Digital Libraries	Citation Counts
Yan <i>et al.</i> [15]	Centrality	Library & Commn. Science	Citation Counts
Bollen <i>et al.</i> [2]	Weighted Page Rank	ISI Journals	Prestige and Status
Dellvale <i>et al.</i> [6]	Weighted Page Rank	Dermatology	Prestige, Status
Leydesdroff [14]	Centrality	Journal Citation Reports	Interdisciplinarity
Gonzalez <i>et al.</i> [9]	Centrality	Mexican Researchers	Research Productivity
Abbasi <i>et al.</i> [1]	Degree Centrality Structural Holes	Library & Info. Science	G-index
Sarigol [24]	Centrality	CS publications	Citations
Newman [22]	Centrality	Medline Physics arXiv Mathematical Rev.	Collaboration Patterns
D'Amour <i>et al.</i> [16]	Centrality	Patents	
Ding [7]	Topic Modeling Path Analysis	Information Retrieval	Topics, Citation

perspective”. *Journal of the American Society for Information Science and Technology*, 62(5):831–845, 2011.

- [11] Knowledge Systems Institute. <http://www.ksi.edu/seke/seke15.html>.
- [12] H. Kretschmer. “Author productivity and geodesic distance in bibliographic co-authorship networks and visibility on the Web”. *Scientometrics*, 60(3):409–420, 2004.
- [13] S. Lee and B. Bozeman. “The impact of research collaboration on scientific productivity”. *Social Studies of Science*, 35(5):673–702, 2005.
- [14] L. Leydesdroff. “Betweenness centrality as an indicator of the interdisciplinarity of scientific journals”. *Journal of the American Society of Information Science and Technology*, 58(9):1303–1319, 2007.
- [15] E. Y. Li, C. H. Liao, and H. R. Yen. “Co-authorship networks and research impact: A social capital perspective”. *Research Policy*, 42:1515–1530, 2013.
- [16] G. C. Li, R. Lai, A. D’Amour, D. M. Doolin, Y. Sun, V. I. Torvik, A. Z. Yu, and L. Fleming. “Disambiguation and co-authorship networks of the U.S. patent inventor database (1975–2010). *Research Policy*, 43(6):941–955, 2013.
- [17] X. Liu, J. Bollen, M. L. Nelson, and H. V. Sompel. “Co-authorship networks in the digital library research community”. *Information Processing and Management*, 41:1462–1480, 2005.
- [18] H. Lu and Y. Feng. “A measure of authors’ centrality in co-authorship networks based on the distribution of collaborative relationships”. *Scientometrics*, 81(2):499–511, 2009.
- [19] T. Martin, B. Ball, B. Karrer, and M. E. J. Newman. “Coauthorship and citation in scientific publishing. *arXiv preprint arXiv:1304.0473*, 2013.
- [20] P. Mutschke. “Mining networks and central entities in digital libraries: A graph theoretic approach applied to co-author networks”. *Advances in Intelligent Data Analysis*, 2810:155–166, 2003.
- [21] M. A. Nascimento, J. Sander, and J. Pound. “Analysis of SIGMOD’s co-authorship graph”. *SIGMOD Record*, 32(3):8–10, 2003.
- [22] M. E. J. Newman. “Coauthorship networks and patterns of scientific collaboration”. *Proc. of the National Academy of the Sciences of the United States of America*, 101(1):5200–5205, April 2001.
- [23] M. E. J. Newman. “The structure of scientific collaboration networks”. *Proc. of the National Academy of Science of the United States of America*, 98(2):404–409, 2001.
- [24] E. Sarigol, R. Pfitzner, I. Scholtes, A. Garas, and F. Schweitzer. “Predicting scientific success based on coauthorship networks. *arXiv:1402.7268*, 2014.
- [25] H. A. Simon. *Administrative Behavior*. Free Press, New York, 1976.

A Rule-based Method for Discovering Trajectory Profiles

Lucas André de Alencar,
Luis Otavio Alvares and Vania Bogorny
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Brasil

Chiara Renso
ISTI-CNR
Pisa, Italy

Alessandra Raffaeta
DAIS - Università Ca'Foscari Venezia
Venice, Italy

Abstract—The discovery of people profiles such as workers, students, families with kids, etc, is of interest for several application domains. For decades, such information has been extracted using census data, and more recently, from social networks, where people's profile is clearly defined. A new type of data that has not been explored for discovering profiles, but which stores the real movement of people, are trajectories of moving objects. In this paper we propose a rule-based method to represent socio-demographic profiles, a moving object history model to summarize the daily movement of individuals, and define similarity functions for matching the profile model and the history model. We evaluate the method for single and multiple profile discovery.

I. INTRODUCTION AND MOTIVATION

The knowledge about people living in a city or country has great value for the public administration as well as for enterprises. To know the population profile may help smart city planners, public transportation administrators, government services or companies to decide if and where to install a new store or to personalize an advertisement, for example. Most attempts to discover and measure the population profiles are through human surveys, and the most well known example is the socio-demographic census with diary activities, periodically done in almost all countries. However, the main drawbacks of the census data are that they: 1) are not up to date since they are usually collected every 5 - 10 years; 2) are expensive to collect, and cover only a small - although statistically significant - part of the population for a short period of time; 3) do not collect the actual movement of the individuals, but only the activities performed during the day and which are mentioned by the user during the interview.

We believe that nowadays we can infer much knowledge and the real behavior about people from their every day movement, about where people really go, when they go, and for how long. We are entering the era of big data, where the real movement behavior of a society can be extracted from its individuals everyday movement. In daily life, in general, we all follow a *routine*, going more or less to the same types of places everyday (e.g. work, gym, supermarket, restaurant, etc). The routine of one person during one week, one month or one year represents the *general pattern of movement* of this person. For instance, a typical routine of a *worker* is to go, in general, four or five times a week to work, while a *student* goes to school/university four or five times a week. On the contrary, an *Unemployed* may have a different routine, as not having a workplace. The routines followed by a similar group of people as the students, workers, or unemployed we call *profiles*.

With the increasing number of GPS trajectory datasets and the definition of semantic trajectories in GPS data [1],

it is possible to infer the real places visited by an object, the duration of the visit, and the frequency of the visits. Based on these visits, it is possible to obtain the routine of an object. An example of semantic trajectory is shown in Fig. 1, where the moving object visits four places (home, university, shopping mall and bar).



Fig. 1. Example of semantic trajectory A.

In the literature of moving object trajectories there are several works for extracting “general patterns” and that summarize the movement of objects, but no works have tried to look deeper into the data to infer more knowledge about the moving object. Only a few works address the discovery of user profiles, but from a different perspective and for different mobility data. For GPS trajectories, which is the focus of this paper, [2] defines as object profile the representative trajectory of a set of similar trips, for car pooling, [3] defines as profiles the users that visit similar places at similar times. [4] is the only work that proposes to infer socio-demographic profiles, but for social network data integrated to GPS trajectories, not only from pure GPS data. In summary, in these works a profile is considered as a set of features which characterize a type of user or a group of users, but not for socio-demographic inference.

In this paper we propose a different perspective. We assume that a description of a mobility behavior for specific socio-demographic categories of users is available and can be represented as “rules”. These rules can be defined by domain experts who describe which is a typical behavior of a specific category (workers, students, unemployed) in a certain application. Another possibility is to run data mining methods on census data or on GPS trajectories to identify groups of users with similar behavior, and label them with the socio-demographic category like “workers” or “students” [5]. Thus, given domain knowledge about how to describe a socio-demographic profile, we propose a *profile model* based on the rules that a moving object should fulfill to belong to a specific profile category. This model allows the user to specify, in a *simple way*, the types of profiles that are interesting for his/her application. How to match GPS trajectories to the profile model is the second focus of this paper, which proposes a moving object history model and a set of similarity functions that are capable to take into account the blurred aspect of

such profiles in two ways: (1) the temporal match is defined considering the overlapping portion between a profile model and the trajectories behavior; (2) the matching function assigns to the match a similarity degree between a profile model and the trajectory behavior. In other words, a trajectory may be matched to several profiles with different similarities, thus being able to discover multiple profiles.

Fig. 2 gives an overview of our proposal. Taking as input GPS trajectories, we first compute the trajectory history model. Then, we compute the similarity between the history model and the profile rules. The output is a set of trajectories labeled with one or more profile names.

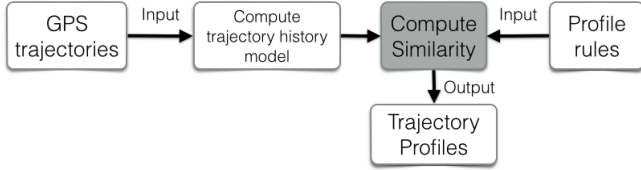


Fig. 2. Overview of the proposal.

The rest of this paper is organized as follows: Section II presents the related work; Section III introduces the basic and new concepts for this work; Section IV presents the algorithm T-Profiles for extracting socio-demographic profiles from trajectories; Section V presents the experimental evaluation of the method with real trajectories; and finally, Section VI presents the conclusion and future work.

II. RELATED WORK

The inference of user profiles from GPS trajectory data, which is the focus of this paper, is very recent, and existing works for GPS trajectory mining have not addressed this problem to extract social information. For instance, [2] defines a set of representative trips performed by the object in his/her historical movement, and a profile is the spatio-temporal trajectory which is frequent in the object's movement history. Profiles in this work are computed for car pooling. Similarly, [6] defines object profile as a sequence of regions frequently visited by the object, and those with similar visits are clustered to infer communities of people. Both previous works focus on raw trajectories, where the object history is a set of space-time points, while we focus on semantic trajectories. [7] proposes a similarity measure that estimates the similarity among semantic trajectories, and the similarity is computed based on the matches of the sequences of categories of visited places between the trajectories. [3] proposes a similarity measure, considering not only the sequences of places but also the travel time to the place and the duration of the visit. Similar to previous works, [8] defines as user profile the mobility pattern of an object, computing the regions of interest (dense regions) and the duration of stays, but does not identify socio-demographic profiles.

In GSM data management, a user profile is defined as his/her mobility pattern [9], [10], [11], extracted from phone calls. Since telecommunication companies have normally a set of information about the user, it becomes trivial to infer user profiles from this type of data. In web logs and social networks, the inference of profiles has been an active area of research.

However, in these networks the user profile is available in the data, while GPS trajectories have only the position and time of the object.

Our work is different from all previous ones since we use a set of rules to describe the behavior of a profile. We also propose a *moving object history model*, which summarizes the individual user movement history in a way that it can be matched with the profile model. As a result, we give the similarity of a user with a given socio-demographic profile or multiple profiles.

III. BASIC DEFINITIONS

Considering that we may infer the profile of people from places they visit, we make use of semantic GPS trajectories and stops [1]. A semantic trajectory A is a sequence of stops $\langle stop_1, \dots, stop_i \rangle$ ordered in time, where each stop is associated to a POI type. Fig. 1 shows an example of semantic trajectory that has four stops $\langle Home, University, ShoppingMall, Bar \rangle$.

Definition 1 (Stop). Let $POIType$ be a type of Point of Interest (POI), $startTime$ and $endTime$ be the start and end time that delimit the interval $[startTime, endTime]$ in which a moving object oid stays at a POI of $POIType$. Then, a stop is a tuple $(oid, POIType, startTime, endTime)$.

In the following section we present the rule-based model (Section III-A), propose a history model (Section III-B), and define similarity measures for matching the rules and the history model (Section III-C).

A. Profile Modeling

A profile is a set of features that represent a group of people with similar characteristics. These characteristics describe a profile/category. For example, the features *go to school, four or five times a week* describe a *student* profile. *Go to work, five times a week*, describe a *worker* profile. These examples of profiles are not mutually exclusive, since a worker can also be a student.

In order to extract socio-demographic profiles from trajectories we define a profile model with features that can be extracted and compared to moving object semantic trajectories. To make the model as simple as possible, we assume that four main features describe a socio-demographic profile: the *type of place* where people go (called $POIType$), *when they go*, *how often* and for *how long* they stay there. With this set of features we define a *profile rule*. We denote with \mathcal{P} the set of profiles we want to investigate.

Definition 2 (Profile rule for a POIType). Let $POIType$ be a type of POI and $p \in \mathcal{P}$ be a profile name. Then a profile rule r for a $POIType$ and a profile p is a tuple of this kind:

$$r = (p, POIType, freq, \omega_f, timeU, weekPeriod, dayPeriod, duration, \omega_d),$$

where $freq$ is the frequency that a $POIType$ is visited in a time unit $timeU$, during certain periods of the day $dayPeriod$, and the period of the week ("weekday", "weekend" or "week") $weekPeriod$, $duration$ is an interval that describes the expected amount of time spent at $POIType$ in the specified period of the day and week. ω_f and ω_d are the weights for

the attributes $freq$ and $duration$, respectively, that should be in the interval $[0,1]$ and their sum must be equal to 1.

An interesting part of this approach is that, to make the rules more expressive, we added a weight ω to the attributes $freq$ and $duration$ to indicate the importance of the attribute to a specific rule.

A rule can express that a specific POIType should not be visited. For instance, a Retired should not have a Workplace. To support this type of profile we allow the definition of positive and negative rules, which are expressed through the attribute frequency. For positive rules the frequency attribute should be above zero ($freq > 0$), and for negative rules $freq = 0$.

If any of the attributes $weekPeriod$, $dayPeriod$ or $duration$ are not relevant, they can be set as Not Applicable (NA). The only exception is the attribute $timeU$, which can only assume NA when the profile rule is negative. Having defined the set of rules for a POIType we can define the profile model, given in Definition 3.

Definition 3 (Profile model). Let $p \in \mathcal{P}$ be a profile name, a profile model for p , called \mathcal{R}^p , is a set of profile rules for POITypes associated with the profile name p .

B. Moving Object History Modeling

The set of all stops of a moving object characterize the *movement history*. This history corresponds to the whole period that the object was tracked (e.g. one week, one month), i.e., the mobility diary. Definition 4 formalizes the object history extracted from semantic trajectories.

Definition 4 (Object History). An object history $h = \langle stop_1, \dots, stop_n \rangle$ is the sequence of stops belonging to the same object such that

$$\forall i \in \{1, \dots, n-1\}, endTime_i \leq startTime_{i+1}$$

where $endTime_i$ and $startTime_i$ refer to the $endTime$ and $startTime$ of the i -th stop of the sequence, respectively.

From the object history, for each place (POIType) visited by an object in his/her trajectories, we compute the values in Definition 5 to summarize the trajectory information.

Definition 5 (Moving Object History Model). Let oid be a moving object identifier and h be its trajectory history. Then, a *moving object history model* for the object history h , called \mathcal{M}_h , is a set of tuples of this kind:

$$m = (oid, POIType, avgFreq, weekPeriod, dayPeriod, avgDuration)$$

where $POIType$ is a type of POI, $avgFreq$ is the average frequency that oid visits $POIType$, $weekPeriod$ specifies when this happens (weekdays, weekends or whole week), $dayPeriod$ indicates the period of the day (morning, afternoon, evening, night) that oid visits $POIType$, and $avgDuration$ is the average amount of time that the object spends at $POIType$ at that weekPeriod and dayPeriod. All these values are extracted from the object history h .

Each tuple $m \in \mathcal{M}_h$ represents the summary of a subset of stops from the object history h with the same $POIType$

for a *weekPeriod* (weekday, weekend and whole week) and *dayPeriod*.

C. Moving Object History Model and Profile Models Matching

As defined in section III-A, there can be two types of profile rules: positive and negative. For each type of rule the matching process is different. In Equation (1) we give the function that computes the similarity between a positive profile rule r and a tuple m of the moving object history model \mathcal{M}_h . It represents the sum of the similarities of frequency and average duration multiplied by their corresponding weight. The tuple m should have the same POIType, weekPeriod and dayPeriod of the ones in the profile rule r , in order to be analyzed.

$$sim_{pos}(m, r) = sim_f \cdot \omega_f + sim_d \cdot \omega_d \quad (1)$$

where $POIType_m = POIType_r$, $weekPeriod_m = weekPeriod_r$, and $dayPeriod_m = dayPeriod_r$.

The similarity functions for frequency sim_f and duration sim_d are defined by functions that follow the same idea of a set membership function in fuzzy logic, and are detailed in section IV, that describes the algorithm.

The similarity for negative rules is defined by Equation (2), where if the $POIType$ defined in the profile rule r is not present in the moving object history model \mathcal{M}_h it has $sim = 1$.

$$sim_{neg}(\mathcal{M}_h, r) = \begin{cases} 1 & \text{if } POIType_r \notin \mathcal{M}_h \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The total similarity between a *moving object history model* \mathcal{M}_h and a profile name $p \in \mathcal{P}$ is given by the function *MATCH* in Equation (3). In general words it is the sum of the similarities of the positive rules sim_{pos} and the sum of the similarities of the negative rules sim_{neg} , divided by the total number of rules of that profile name p .

$$MATCH(\mathcal{M}_h, p) = \frac{\sum sim_{pos}(m_i, r_j) + \sum sim_{neg}(\mathcal{M}_h, r_k)}{|\mathcal{R}^p|} \quad (3)$$

where \mathcal{R}^p is the set of rules of the profile name p

In the following section we present the algorithm T-Profiles, to extract socio-demographic profiles from trajectories.

IV. T-PROFILES: AN ALGORITHM FOR DISCOVERING TRAJECTORY PROFILES

Listing 1 shows the pseudo-code of the proposed algorithm to extract profiles from trajectory data, named T-Profiles. The algorithm receives as input a set of semantic trajectories T , a set of profile names \mathcal{P} , a set of profile models \mathcal{R} , and the minimal similarity degree ϵ for an object to be considered similar to a profile name. The output is a set of moving objects labeled with a profile name p and the similarity degree between the object and the profile.

The first step is to compute the history model for each moving object in T (lines 12, 13). We summarize the trajectories

to the structure of Definition 5, for each visited place. Once computed, the moving object history model will be compared with all rules of each profile name p in the profile model \mathcal{R}^p (lines 14-38).

The similarity of positive rules (line 18) is computed according to Equation (1), with sim_f defined by Equation (4) and sim_d defined by Equation (5), presented in the following. The similarity of negative rules (line 24) is calculated according to Equation (2). These values are used to compute the matching between a moving object history model and a profile name (line 33).

If one or more of the negative rules of a profile name are not satisfied by the object history model, the similarity is set to zero, since the negative rules are mandatory (line 31). In case the total similarity is greater than the threshold ϵ , the moving object identifier, the profile name and the similarity degree are added to the output set ψ of trajectory profiles (line 36). This step finishes the analysis of one profile name and the algorithm returns to line 14 to test the next profile name in \mathcal{P} with the current object history. Notice that the algorithm has the capability to return multiple profiles, i.e., a moving object can belong to several profiles in case the match is above ϵ .

Listing 1. Pseudo-code of the algorithm T-Profiles

```

1  Input:  $T$  // set of semantic Trajectories
2   $\mathcal{P}$  // set of profile names
3   $\mathcal{R}$  // profile models  $\mathcal{R} = \cup_{p \in \mathcal{P}} \mathcal{R}^p$ 
4   $\epsilon$  // minimal similarity degree for
5  // an object belonging to a profile
6
7  Output:  $\psi$  // set of moving object profiles
8
9  Method:
10
11   $\psi = \{\}$  // empty set
12  for each moving object history  $h \in T$  do
13     $\mathcal{M}_h = \text{buildMovingObjectHistoryModel}(h)$ 
14    for each profile name  $p \in \mathcal{P}$  do
15      sumPos = 0
16      for each positive rule  $r \in \mathcal{R}^p$  do
17        for each  $m \in \mathcal{M}_h$  do
18          sumPos = sumPos +  $sim_{pos}(m, r)$ 
19        end for
20      end for
21      negativeRulesNotHold = False
22      sumNeg = 0
23      for each negative rule  $r \in \mathcal{R}^p$  do
24        aux =  $sim_{neg}(\mathcal{M}_h, r)$ 
25        sumNeg = sumNeg + aux
26        if aux = 0
27          negativeRulesNotHold = True
28        end if
29      end for
30      if negativeRulesNotHold
31        MATCH = 0.0
32      else
33        MATCH = (sumPos + sumNeg) /  $|\mathcal{R}^p|$ 
34      end if
35      if MATCH >  $\epsilon$ 
36         $\psi$ .add( $h.oid, p, MATCH$ )
37      end if
38    end for
39  end for
40  return  $\psi$ 

```

Similarity functions

The frequency and the duration similarity can be implemented in different ways. After performing several experiments we implemented the following in T-Profiles: the similarity for frequency (sim_f) is defined by Equation (4),

where $avgFreq_m$ is the average frequency computed in the history model tuple m , $freq_r$ and $timeU_r$ are respectively, the frequency and the time unit defined in the profile rule r . The function $days(timeU_r)$ returns the number of days that the time unit represents (e.g. if $timeU_r = week$, then $days(timeU_r)$ returns 7). When $avgFreq_m = 0$, means that the object did not visit the POIType, so $sim_f = 0$. When $avgFreq_m$ is lower than the frequency defined in the profile rules (represented by $\frac{freq_r}{days(timeU_r)}$), then sim_f increases linearly from 0 to 1. If $avgFreq_m$ is greater than the frequency defined in the profile rules, then $sim_f = 1$.

$$sim_f = \begin{cases} 0 & \text{if } avgFreq_m = 0 \\ \frac{avgFreq_m \cdot days(timeU_r)}{freq_r} & \text{if } avgFreq_m < \frac{freq_r}{days(timeU_r)} \\ 1 & \text{if } avgFreq_m \geq \frac{freq_r}{days(timeU_r)} \end{cases} \quad (4)$$

The duration similarity (sim_d) is defined by Equation (5), and is illustrated in Fig. 3 for $duration_r$ defined as the interval [1:00, 2:00]. Fig. 3 shows that an $avgDuration_m$ between 1 and 2 hours will have $sim_d = 1$. For an $avgDuration_m$ between 0.5 hour and 1 hour the similarity increases linearly from 0 to 1, and for an $avgDuration_m$ between 2 and 2.5 hours the similarity decreases linearly from 1 to 0.

$$sim_d = \begin{cases} 0 & \text{if } avgDuration_m < minGlobal \vee \\ & avgDuration_m > maxGlobal \\ 1 - \frac{minDur - avgDuration_m}{minDur - minGlobal} & \text{if } minGlobal < avgDuration_m \wedge \\ & avgDuration_m < minDur \\ 1 & \text{if } minDur \leq avgDuration_m \wedge \\ & avgDuration_m \leq maxDur \\ 1 + \frac{maxDur - avgDuration_m}{maxGlobal - maxDur} & \text{if } maxGlobal > avgDuration_m \wedge \\ & avgDuration_m > maxDur \end{cases} \quad (5)$$

where

$$duration_r = [minDur, maxDur] \\ minGlobal = minDur - (minDur * 0.5) \\ maxGlobal = maxDur + (minDur * 0.5)$$

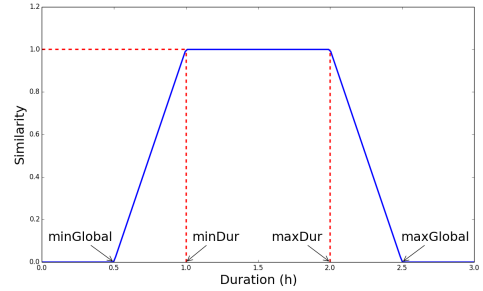


Fig. 3. Duration similarity function sim_d for $duration_r = [1:00 - 2:00]$.

V. EXPERIMENTAL EVALUATION

We evaluate our proposal using two datasets, a trajectory set built from census data where we have the ground truth (V-A), and a GPS trajectory dataset of car trajectories collected in Florence, Italy (V-B).

A. Census Trajectories

As it is still very difficult to obtain a dataset of semantic trajectories with a ground truth, we first evaluate the algorithm T-Profiles on a “trajectory” dataset generated from census data, where we have the ground truth. This dataset is a census of

activity diaries collected in Italy in 2008, having the socio-demographic profile of each individual that was interviewed. Each activity diary corresponds to the activities of one person during one day, and can be seen as the “semantic trajectories” of each individual, because they contain the place of activity (that corresponds to the POIType of the stops), the activities performed at the place, and the begin and end time of the activities. Examples are: sleeping at home from 10PM to 8AM, profile *retired*; working at a workplace from 10AM to 5PM, profile *worker*; studying at the university from 2PM to 6PM, profile *student*, etc. The most significant profiles in the database are: *worker*, *retired*, *unemployed*, *housewife with kids* and *student*.

As one day of activities is not enough to determine the profile of a person, we preprocessed the data grouping diaries that belong to the same socio-demographic profile, considering 14 days of activities. As a result, we obtained trajectories of 14 days long for 829 objects. The amount of objects for each profile is shown in the second column of Table II.

TABLE I. PROFILE RULES.

Profile Name p	POIType	freq (ω_f)	timeUnit	weekPeriod	dayPeriod	duration (ω_d)
Worker	Workplace	4 (0.8)	week	NA	NA	03:00 - 09:00 (0.2)
Student	School/Univ.	3 (0.5)	week	weekday	NA	03:00 - 06:00 (0.5)
Retired	Workplace	0 (1)	NA	NA	NA	NA
Retired	School/Univ.	0 (1)	NA	NA	NA	NA
Retired	Bar	1 (1)	week	NA	Morn, Aftn	NA
Unemployed	Workplace	0 (1)	NA	NA	NA	NA
Unemployed	School/Univ.	0 (1)	NA	NA	NA	NA
Unemployed	Bar	1 (1)	week	NA	Evening	NA
Unemployed	Restaurant	1 (1)	month	NA	NA	NA
Unemployed	Sport court	2 (1)	month	NA	NA	NA
Housewife Kids	School/Univ.	3 (0.5)	month	weekday	NA	00:20 - 01:00 (0.5)
Housewife Kids	Commercial estab.	3 (1)	week	weekday	NA	NA

Table I shows the rules considered in this experiment. A *worker* is identified by the POIType Workplace, that should be visited with a frequency of 4 times a week with duration between 3 and 9 hours. We define a broad range for duration to obtain all types of workers (full time and part time). Notice that we defined a higher weight for the frequency (0.8), because this attribute is more important than the duration.

The rule for the profile named *Student* expresses that this profile should visit a POIType related to educational institutions, such as schools or universities, for at least 3 times a week on weekdays, with a duration between 3 and 6 hours per day, to include full time and part time students. The weights for the attributes *freq* and *duration* are both 0.5.

For the profile *Retired*, we defined two negative rules related to workplace and educational places, to distinguish between workers and students, since it is expected that most retired do not have a workplace and do not go to school. However, these rules are not enough to distinguish a retired from an unemployed. Then, as they are supposed to go more often to bars or cafes, we create a rule with this kind of POIType, in the period of morning and afternoon, i.e., during the day.

The profile *Unemployed* may have similar behavior to the *Retired*, having no working place and not going to school to distinguish these profiles. To distinguish an unemployed from a retired we define three positive rules: POIType Bar visited during the evening, POIType Restaurant visited only once a month, and visits to sport places.

A housewife that has children can be identified if the person

visits educational places such as schools. But the difference from the profile student is the frequency and the duration. The profile does not need to go every day to take the child to the school, but should at least visit a POIType school sometimes to express that there is a relationship with educational place. Defining a rule forcing a housewife with kids to go very frequently to educational places would limit the discovery only of cases where the housewife takes the kids to school everyday.

Table II shows the results for similarity ϵ of 60%, 70% and 80%. For similarity 70%, for instance, T-Profiles detected 478 workers out of 479, and 73 out of 74 students. For the profile Housewife Kids, 24 instances were discovered. The most difficult classification is to distinguish unemployed and retired, because their behavior is very similar, but still 158 retired from 224 were detected.

TABLE II. PROFILES FOR 60%, 70% AND 80% SIMILARITY.

Profiles	Total	$\epsilon = 0.6$	$\epsilon = 0.7$	$\epsilon = 0.8$
Worker	479	479	478	473
Housewife Kids	35	28	24	20
Unemployed	17	9	9	9
Retired	224	185	158	158
Student	74	74	73	72

Table III shows the precision and recall for each profile, considering the similarities for each profile name as well as the average for all objects. T-Profiles shows a very high average precision, about 97%. The recall is also high, between 88% and 93% with these values of ϵ .

TABLE III. PRECISION AND RECALL

	$\epsilon = 0.6$		$\epsilon = 0.7$		$\epsilon = 0.8$	
	Precision	Recall	Precision	Recall	Precision	Recall
Worker	1.000	1.000	1.000	0.997	1.000	0.987
Housewife Kids	0.583	0.800	0.750	0.685	0.769	0.571
Unemployed	0.529	0.529	0.529	0.529	0.600	0.529
Retired	0.953	0.825	0.957	0.705	0.957	0.705
Student	1.000	1.000	1.000	0.986	1.000	0.972
Avg.	0.960	0.934	0.968	0.895	0.970	0.882
Avg. F1 measure	0.946		0.926		0.921	

In these results we considered only the highest similarity for each profile name, but we can also analyze all similarities that are above the threshold ϵ , having multiple profiles. Table IV shows some examples of the output of T-Profiles. Each row corresponds to an object. The multiple profile column shows all profile names that have similarity above 80%. For the object 842, for instance, the similarity with Worker and Housewife Kids is above 90%, so this object is labeled as Worker and Housewife Kids.

TABLE IV. MULTIPLE PROFILES FOUND USING $\epsilon = 0.8$

oid	Worker	Housewife Kids	Unemployed	Retired	Student	Multiple profile
16	0.000	0.174	0.800	0.980	0.000	Retired, Unemployed
131	0.000	0.261	0.800	0.980	0.000	Retired, Unemployed
842	1.000	0.949	0.000	0.000	0.179	Worker, Housewife Kids
600	1.000	0.897	0.000	0.000	0.100	Worker, Housewife Kids

B. Florence Dataset

The Florence dataset is a sample of car trajectories collected by an insurance company during one month, but the average tracking period of one object was 10 days. From this dataset, we selected all trajectories with more than 10 stops, labeled with their POI Types and with at least 10 days history.

Here we show the flexibility of T-Profiles, where the user can choose any level of profile category analysis, from the more general to the more detailed. We are interested in Full Time Workers, Part Time Workers, Weekend Workers and Night Workers. Considering the rules defined in Table I, we extended the rules set with new profiles of workers. Table V shows the rules for workers, where the duration distinguishes Full Time and Part Time Workers; while the frequency, week period, and day period distinguish Weekend and Night Workers.

TABLE V. PROFILE RULES FOR WORKER PROFILES.

Profile Name p	POI Type	freq (ω_f)	timeUnit	weekPeriod	dayPeriod	duration (ω_d)
Full Time Worker	Workplace	4 (0.5)	week	NA	NA	07:00 - 09:00 (0.5)
Part Time Worker	Workplace	4 (0.5)	week	NA	NA	03:00 - 05:00 (0.5)
Weekend Worker	Workplace	1 (1)	week	weekend	NA	NA
Night Worker	Workplace	3 (1)	week	NA	Evening, Night	NA

Table VI shows the result for $\epsilon = 0.8$, where T-Profiles labeled 36 Full Time Workers, 16 Part Time Workers, 31 Weekend Workers, 21 Night Workers, 4 Students, and 2 Retired.

TABLE VI. PROFILES FOR 70%, 80% AND 90% SIMILARITY.

Profiles	$\epsilon = 0.7$	$\epsilon = 0.8$	$\epsilon = 0.9$
Full Time Worker	56	36	32
Part Time Worker	26	16	9
Weekend Worker	37	31	31
Night Worker	24	21	15
Student	4	4	4
Retired	2	2	0
Unemployed	1	0	0
Housewife Kids	0	0	0

Table VII shows some examples of the output of T-Profiles, with some single and multiple profiles. Notice that objects 893757 and 820817 were labeled with multiple profiles. For oid 893757 the similarity degree was 100% with Part Time and Weekend Worker, while object 820817 had similarity of 100% with Full Time and Night Worker. Objects 85000 and 288807 had similarity above 80% with the profile category Full Time and Night Worker, respectively. Another example is the object with oid 757727, that had similarity 100% with the profile Student.

VI. CONCLUSION AND FUTURE WORK

In this paper we made a first attempt to go deeper in the analysis of moving object trajectories, analyzing every individual object mobility history in order to discover the socio-

TABLE VII. EXAMPLES OF THE OUTPUT OF T-PROFILES FOR $\epsilon = 0.8$

oid	Housewife Kids	Unemployed	Retired	Student	Full Time Worker	Part Time Worker	Weekend Worker	Night Worker	Profile
893757	0.000	0.000	0.000	0.000	0.603	1.000	1.000	0.000	Part Time Worker, Weekend Worker
820817	0.000	0.000	0.000	0.000	1.000	0.500	0.000	1.000	Full Time Worker, Night Worker
85000	0.326	0.000	0.000	0.106	0.977	0.477	0.000	0.000	Full Time Worker
288807	0.000	0.000	0.000	0.000	0.743	0.318	0.636	0.848	Night Worker
757727	0.250	0.000	0.000	1.000	0.000	0.000	0.000	0.000	Student
1255063	0.000	0.750	0.852	0.000	0.000	0.000	0.000	0.000	Retired

demographic status of each individual. While the discovery of socio-demographic profiles is very trivial in social networks, GSM calls and weblog data, where far more information is available, the discovery of socio-demographic profiles from GPS trajectories is a challenge. In this paper we proposed a profile model, as a set of very simple rules that the user can express to discover any type of profile. We also introduced a moving object history model that summarizes the historical traces of moving objects, that is independent of a specific profile model. Finally, we proposed a matching process that provides the similarity between a given profile name and a moving object based on his/her trajectory summary. As future work, we will go deeper in the analysis to discover more complex profiles such as gender, marital status, income, etc.

ACKNOWLEDGMENT

This work was supported by EU project FP7- PEOPLE SEEK (N.295179 <http://www.seek-project.eu>) and the Brazilian agencies CAPES and CNPQ.

REFERENCES

- [1] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macedo, B. Moelans, and A. Vaisman, "A model for enriching trajectories with semantic geographical information," in *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*. ACM, 2007.
- [2] R. Trasarti, F. Pinelli, M. Nanni, and F. Giannotti, "Mining mobility user profiles for car pooling," in *Proceedings of the 17th ACM SIGKDD*, ser. KDD '11. ACM, 2011, pp. 1190–1198.
- [3] X. Xiao, Y. Zheng, Q. Luo, and X. Xie, "Finding similar users using category-based location history," in *Proceedings of the 18th SIGSPATIAL*. ACM, 2010, pp. 442–445.
- [4] V. W. Zheng, Y. Zheng, and Q. Yang, "Joint learning user's activities and profiles from GPS data," in *Proceedings of the 2009 LBSN*, ser. LBSN '09. New York, NY, USA: ACM, 2009, pp. 17–20.
- [5] S. Jiang, J. Ferreira, and M. C. González, "Clustering daily patterns of human activities in the city," *Data Mining and Knowledge Discovery*, vol. 25, no. 3, pp. 478–510, 2012.
- [6] C.-C. Hung, C.-W. Chang, and W.-C. Peng, "Mining trajectory profiles for discovering user communities," in *Proceedings of the 2009 LBSN*, ser. LBSN '09. New York, NY, USA: ACM, 2009, pp. 1–8.
- [7] J. J.-C. Ying, E. H.-C. Lu, W.-C. Lee, T.-C. Weng, and V. S. Tseng, "Mining user similarity from semantic trajectories," in *Proceedings of the 2nd ACM SIGSPATIAL LBSN*, ser. LBSN '10. New York, NY, USA: ACM, 2010, pp. 19–26.
- [8] X. Chen, J. Pang, and R. Xue, "Constructing and comparing user mobility profiles," *ACM Transactions on the Web (TWEB)*, vol. 8, no. 4, p. 21, 2014.
- [9] M. A. Bayir, M. Demirbas, and N. Eagle, "Mobility profiler: A framework for discovering mobility profiles of cell phone users," *Pervasive and Mobile Computing*, vol. 6, no. 4, pp. 435 – 454, 2010.
- [10] B. Furlletti, L. Gabrielli, C. Renso, and S. Rinzivillo, "Analysis of GSM calls data for understanding user mobility behavior," in *2013 IEEE International Conference on Big Data*, Santa Clara, California, 2013, pp. 550–555.
- [11] M. Dash, H. L. Nguyen, C. Hong, G. E. Yap, M. N. Nguyen, X. Li, S. Krishnaswamy, J. Decraene, S. Antonatos, Y. Wang, D. T. Anh, and A. Shi-Nash, "Home and work place prediction for urban planning using mobile network data," in *IEEE 15th MDM*, vol. 2, July 2014, pp. 37–42.

A Balanced Method for Budgeted Influence Maximization

Xinhui Xu*, Yong Zhang[†], Qingcheng Hu*, Chao Li[†], Chunxiao Xing[†]

Department of Compute Science and Technology

Tsinghua National Laboratory for Information Science and Technology

Research Institute of Information Technology, Tsinghua University, Beijing, China

* {xuxh13, hqc10}@mails.tsinghua.edu.cn

[†] {zhangyong05, li-chao, xingcx}@mail.tsinghua.edu.cn

Abstract—With the flourish of Web-based large Online Social Networks (OSNs), people on OSNs can easily yield influence on others. Finding how the influence spreads and maximizing influence spread within OSNs have been extensively studied. State-of-the-art researches suffer two defects: (a) need to acquire the topological structure of the network, which is impractical for the continuously changing networks in real life and thus can not balance very well between influence spread and running time; (b) assign the same cost for every node in OSN which cannot reflect the reality. To solve these problems we firstly propose PageRank Based Cost (PRBC) model to assess the cost of nodes in OSN according to their importance (influence); secondly we present Budgeted Random Maximal Degree Neighbor (BRMDN) algorithm by exploiting the scale free property. Results from extensive experiments show that BRMDN can well balance influence spread and running time.

Keywords-social network; cost model; influence maximization;

I. INTRODUCTION

A. Background and Motivations

The Web today is a growing universe of interlinked Web pages and Web applications, teeming with videos, photos, and interactive content. The Web-based large online social network(OSN)s such as Facebook, Twitter, WeChat etc. acquire great success for their interactive features like sharing, forwarding and discussing contents. According to eBizMBA, in February 2015, Facebook has estimated unique monthly visitors numbered 900 million¹. People spend a lot of time on these communication platforms making friends, sharing daily affairs, spreading interesting news, and expressing different opinions, which provides us with affluent real-life data to mine valuable information. Taking advantage of the popularity of OSNs, many researchers have studied diffusion phenomenon in OSNs, which includes the diffusion of news, ideas, innovations, and the adoption of new products [1]. These diffusion phenomenons are referred as influence diffusion or propagation in [1]. Influence maximization is an extensively investigated topic in influence diffusion [1] [2] [3] [4]. It tries to find a set of nodes in one OSN to maximal the influence spread over the OSN under certain diffusion model such as Independent Cascade (IC) proposed in [2].

However, the aforementioned research works suffer two defects: (a) need to acquire the topological structure of the network, which is impractical for the continuously changing networks in real life and thus can not balance very well between influence spread and running time; (b) assign the same cost for every nodes in OSN which cannot reflect the real situation in life. In reality, time can be a critical factor in some situations such as disease controlling, emergency evacuation. Furthermore, different nodes in one OSN should not be assigned the same cost. For example, in the domain of online advertisement, different service providers have different advertising prices [5].

Taking account of the above situations, also with the fact that most of the large OSNs are complex networks and have the scale free property [6], we firstly define a cost function to assess the cost of a given node in one OSN, then research budgeted influence maximization problem and propose our algorithm.

B. Our Contributions

In this paper, we propose a cost function to assess every node in terms of their influence and dedicate to solving the problem of budgeted influence maximization. Our contributions in this paper are summarized as follows.

- We propose PRBC model to assess nodes' costs according to their importances (influences) which are assessed by the nodes' PageRank value [7] and degrees in OSN.
- We exploit the scale-free property [6] [8] that most OSNs hold, and then propose BRMDN under PRBC model.
- Finally, we test the performance of BRMDN under two real datasets with extensive experiments, which proves the effectiveness and efficiency of the proposed algorithm.

The rest of this paper is organized as follows. Section II introduces the related work. Section III provides preliminaries for budgeted influence maximization. Section IV presents our algorithms and the theoretical analysis. Section V shows our experimental results. Finally we conclude the paper in section VI.

¹<http://www.ebizmba.com/articles/social-networking-websites>

II. RELATED WORK

A. Diffusion Models

In influence maximization process, we label every node with one status—activated or inactivated. We tag a node *activated* if it accepts the message or event we concern, *inactivated* otherwise. In a widely connected network, people may influence each other by publishing, sharing, re-directing messages or news etc. Therefore, influence spreading is to a large extent similar to information diffusion process. One pioneer information diffusion model is IC [2].

IC is firstly proposed by D Kempe et al. in [2], and now becomes the most important model in influence maximization problem. Given OSN $G = (V, E)$, an initial seed set S , let $p(u, v)$ denote the probability of u influencing v and $p(u, v)$ is independent assigned for $\forall u, v \in V$. IC runs as follows: Let S_t denote the nodes activated in step (time) t , t_e represents the step when the activation process ends, initially we have $t = 0, S_t = S$. At step $t + 1$, every node $u \in S_t$ tries to activate its out-neighbors $v \in V \setminus \bigcup_{0 \leq i \leq t} S_i$ with an independent probability of $p(u, v)$. This procedure proceeds until no more nodes can be activated. It should be mentioned that each node can activate its out-neighbors one time and when a node is activated, it never fails. The final activated set can be calculated by $\bigcup_{0 \leq i \leq t_e} S_i$ which we denote as $\sigma_{IC}(S)$.

B. Influence Maximization

1) *Greedy Algorithms*: In [2], D Kempe et al. proposed a general greedy algorithm for influence maximization and proved that general greedy algorithm can approximate to the optimal solution by a factor of $1 - 1/e$, but it is extremely expensive to compute. CELF [9] improved general greedy algorithm by exploiting the property of submodular function. Results show that CELF achieves 700 times faster than general greedy algorithm. NewGreedy and MixGreedy in [10] presented two variants of greedy algorithm which aim at improving effectiveness and efficiency, yet still suffer high computation cost.

2) *Random Based Algorithm*: Random algorithm always contains a randomization procedure within itself, in the field of influence maximization, the most obvious and straightforward random based algorithm is referred as Random in [2] [10]. It runs as follows, it iterates for k times, every time it randomly selects a node which has not been added to the seed set. As we can see from Table I, Random runs fastest, but we will show later it performs the worst in terms of influence spread.

3) *Degree Based Heuristic Algorithms*: In OSNs, one node that has a larger number of in-neighbors most probably means that it is more important. It is referred as degree centrality [11]. DegreeHeuristic algorithm sorts all the nodes according to their degree and then chooses the Top-K nodes. In [10] SingleDiscount is a simple degree discount heuristic

Table I
TIME COMPLEXITY OF ALGORITHMS

Algorithms	Complexity
Random	$O(K)$
Degree Heuristic	$O(m)$
Degree Discount	$O(K * \log(n) + m)$
Single Discount	$O(K * \log(n) + m)$
New Greedy IC	$O(KRm)$
CELF Greedy	$O(KnRm/700+)$
General Greedy	$O(KnRm)$

where each neighbor of a newly selected seed discounts its degree by one, while DegreeDiscount is a more accurate degree discount heuristic algorithm. It excludes nodes that can possibly be influenced by nodes which have already been added into the seed set. Experiments show that SingleDiscount and DegreeDiscount have almost the same result in influence spreading as greedy algorithms, while achieving significant speedup in running time.

The time complexities of some aforementioned algorithms are shown in Table I, where K denotes # of nodes in the initial seed set, n represents # of vertices, m depicts # of edges in the given graph and R is # of rounds in the specific algorithm. These notations used in the following sections will have the same meaning if there is no explicit declaration.

C. Cost Models

Most state-of-the-art influence maximization algorithms such as [1] [2] [3] [4] [10] etc. take the constant cost (i.e. unit cost) model. That is, $\mathcal{CF}(s) = 1$ for all node $s \in V$, $\mathcal{CF}(\cdot)$ denotes cost function. In [9], J. Leskovec et al used # of posts for their case 1, they assigned a non-negative cost for case 2; [5] does not give their detailed cost model; while [12] randomly gives a value to nodes in OSN. To our best of knowledge, few non-constant cost models have been used for budgeted influence maximization.

III. PRELIMINARIES

A. Scale-free Networks

In the real world, there are numerous networks existing in form of complex network [6] such as biosphere, citation network, OSNs and so on. Many large OSNs (such as Facebook, Twitter, MySpace, Flickr [13]) share the property of scale-free [6]. A node in a network with degree k subjects to power-law distribution has probability $p(k) = ck^{-\gamma}$ [14]. When the power-law distribution exponent γ values are between 2 and 3, the network holds the property of scale-free.

B. PageRank Algorithm & Proposed Cost Model

PageRank algorithm has been widely taken as a method for measuring the importance of web pages which was firstly proposed in [7]. With the web pages modeled as nodes, and hyperlinks between them represented as edges, the interlinked Web can be seen as a complicated graph. As

we have described in previous section, OSN is represented by $G = (V, E)$, so we can apply the PageRank algorithm to the OSN to estimate the importance of nodes. According to nodes' differences in importance we give them different cost. The cost model is shown as follows.

Definition 1. (*PageRank Based Cost*) Given an OSN $G = (V, E)$, a pre-defined increase factor δ and coefficient λ , for $\forall u \in V$, we define its PageRank Based Cost, $PRBC(u)$, as follows,

$$PRBC(u) = \frac{\lambda(PR(u) + \delta)\mathcal{D}(u)}{\mathcal{D}(v_{max})}, v_{max} = \operatorname{argmax}_{v \in \mathcal{N}(u) \cup \{u\}} \mathcal{D}(v)$$

where $PR(u)$ is the PageRank value of node u , $\mathcal{D}(u)$ is the degree of node u , $\mathcal{N}(u)$ is the neighbor set of node u .

C. Non-Constant Cost Influence Maximization

In reality, the nodes in OSN have different influences should be assigned different costs. Compared with normal unit cost influence maximization problem, Non-Constant Cost Influence Maximization (NCC-IM) has tighter constraints. We formally formulate it as follows,

Definition 2. (*NCC-IM*) Given OSN $G = (V, E)$, a constant K , a cost function $\mathcal{CF}(\cdot)$, and a budget limit \mathcal{B} , finding a set S with K nodes which subjects to the following constraints:

$$\sigma(S) = \operatorname{argmax}_{|S| \leq K \wedge S \subseteq V} \operatorname{Inf}(S), \sum_{u \in S} \mathcal{CF}(u) < \mathcal{B}$$

where $\sigma(S)$ denotes the expected nodes set finally activated by giving an initial set S under influence spread function Inf .

Noting that in NCC-IM problem, we have constraints that $|S| \leq K$, because that in our algorithm and other algorithms we implemented as baselines, we find that when $\sum_{u \in S} \mathcal{CF}(u) > \mathcal{B}$, all the algorithms should stop.

IV. BUDGETED INFLUENCE MAXIMIZATION ALGORITHM

Finding exact maximal influence spread is a NP-Hard problem and the improved greedy algorithm variants are expensive to compute. Random algorithms randomly choosing K nodes also perform arbitrarily bad. [8] [15] [16] find that in many OSNs vertices' connectivities follow scale-free property that most vertices are sparsely connected. However, a small number of vertices are densely connected. We design our own algorithm by combining randomly heuristic method with the properties of scale-free networks.

A. Algorithm Design

Enlightened by the random algorithm and the property of scale-free network, we propose Budgeted Random Maximal Degree Neighbor (BRMDN) algorithm. Firstly, BRMDN randomly selects a node u in the network; then it uses the algorithm MDN (Algorithm 1) to find the candidate node

which will be added to initial set S if it satisfies the budget constraints. we repeatedly run the above procedure to get the qualified S . We give BRMDN as Algorithm 2.

Algorithm 1 MDN: Find the candidate node

Input: $G = (V, E)$; node u ; exclusive set \mathcal{ES} ;

Output: Node v_{max} selected to be added to the seed set

```

1:  $v_{max} \leftarrow u$ ;  $v_{degree} \leftarrow \mathcal{D}(u)$ ;
2: for all  $nbr \in \mathcal{N}(u)$  do
3:   if  $nbr \notin \mathcal{ES} \wedge v_{degree} < \mathcal{D}(nbr)$  then
4:      $v_{degree} \leftarrow \mathcal{D}(nbr)$ ;
5:      $v_{max} \leftarrow nbr$ ;
6:   end if
7: end for
8: return  $v_{max}$ 

```

Algorithm 2 BRMDN: Compute the seed set

Input: $G = (V, E)$; seed set size K ; budget limit \mathcal{B} ; fault tolerance factor τ ; cost function \mathcal{CF} ;

Output: Seed set S with $|S| \leq K \wedge \sum_{s \in S} \mathcal{CF}(s) < \mathcal{B}$

```

1:  $S \leftarrow \Phi$ ;  $i \leftarrow 0$ ;  $totalcost \leftarrow 0$ 
2: while  $i < K$  do
3:   Randomly select a node  $u \in V \setminus S$ 
4:   Select  $u_{max} \leftarrow MDN(G, u, S)$ 
5:   if  $\mathcal{CF}(u_{max}) + totalcost < \mathcal{B} + \tau$  then
6:      $S \leftarrow S \cup \{u_{max}\}$ 
7:      $totalcost \leftarrow totalcost + \mathcal{CF}(u_{max})$ 
8:      $i \leftarrow i + 1$ 
9:   if  $totalcost > \mathcal{B}$  then
10:     $i \leftarrow K + 1$ 
11:   end if
12: end if
13: end while
14: return  $S$ 

```

Noting that through MDN procedure in BRMDN, we only need to know the local topological structure around node u , which can greatly boost computing efficiency.

B. Feasibility of BRMDN

For network $G = (V, E)$ which subjects to the power-law distribution, a node with degree k has probability $p(k) = ck^{-\gamma}$. Let k_{max} be # of the maximal degree and k_{min} be the minimal one, so we have:

$$\int_{k_{min}}^{+\infty} p(k)dk = \frac{1}{n}, \int_{k_{min}}^{+\infty} p(k)dk = 1 \quad (1)$$

By solving (1), we get, $k_{max} = k_{min}n^{\frac{1}{\gamma-1}}$. For arbitrary edge which starts from node u , let p_{Top-K} denote the probability of connecting a node with degree great than or equal to Top-K (such node is also known as one Hub of the

network). We can get

$$p_{Top-K} = \int_{k_{Top-K}}^{k_{max}} p(m)dm = \frac{k_{max}^{2-\gamma} - k_{Top-K}^{2-\gamma}}{k_{max}^{2-\gamma} - k_{min}^{2-\gamma}} \quad (2)$$

If the seed set size is K , the probability to get at least one hub node is $p_{hub} = 1 - (1 - p_{Top-K})^K - \epsilon$. Where ϵ denotes the budget \mathcal{B} affects the p_{hub} , actually if $\mathcal{B} > K\zeta$ then $\epsilon = 0$, where $\zeta = \max_{u \in V} \{\mathcal{CF}(u)\}$ is the highest cost in the OSN. If K is bigger enough (say $K = 30$) then we can have $(1 - p_{Top-K})^K \rightarrow 0$, and finally $p_{hub} \approx 1 - \epsilon$. Here ϵ is a random factor which is determined by random process in BRMDN. But we can alleviate the random disturbance by iterating BRMDN many times to decrease constraint ϵ to a small number and so p_{hub} is close to 1. A high value of p_{hub} indicates that BRMDN with a large possibility can have at least one hub node in initial set S .

C. Time Complexity

According to Algorithm 2, after a node is randomly chosen, we have to traverse its all neighbors. Let \bar{k} denote the average degree in a scale-free network, we have:

$$\bar{k} = \sum_1^n kp(k) = \sum_1^n kck^{-\gamma} = c \sum_1^n \frac{1}{k^{\gamma-1}}, p(k) = ck^{-\gamma} \quad (3)$$

Lemma 1. Let $G = (V, E)$ be a network that holds scale-free property, if there are no self-loops or multiple links between two nodes in G , there does not exist a G with $1 < \gamma < 2$.

Proof: From section IV-B, we already have $k_{max} = k_{min}n^{\frac{1}{\gamma-1}}$. Now assume that there exists a network that subjects to power-law distribution and has $1 < \gamma < 2$, then we can get $0 < \gamma - 1 < 1$, $n^{\frac{1}{\gamma-1}} > n$, and $k_{max} = k_{min}n^{\frac{1}{\gamma-1}} > n$, which means that a node with the largest degree is even bigger than the number of the total nodes in network G . It contradicts the fact that n is the total # of nodes of network G . ■

According to lemma 1, $\gamma > 2$. Then

$$\bar{k} = c \sum_1^n \frac{1}{k^{\gamma-1}} \leq c \sum_1^n \frac{1}{k} = c \ln(n), n \rightarrow +\infty \quad (4)$$

If the size of the initial seed set is K , the time complexity of Algorithm 2 can be computed by $K\bar{k}$. By substituting \bar{k} with equation (4), we get $O(K \log(n))$.

V. EXPERIMENTS

Considering different networks with different topological structures, we choose two datasets which subject to power-law distribution and have different γ values (Table II). We will show that BRMDN performs very well in terms of both influence spread and running time.

Table II
STATISTICS OF TWO REAL NETWORKS

Dataset	n	m	\bar{k}	k_{max}	γ
Blogs	3982	6803	3.42	189	2.453
Facebook	4039	88234	43.69	1045	2.509

A. Experimental Setup

The two real scale-free networks listed in Table II are summarized as follows:

- **Blogs [17].** It contains about 4K nodes and 6K edges. Obviously this network is sparsely connected, and $\frac{\#edges}{\#nodes} = 1.70$.
- **Facebook [18].** It is just a small part of users of Facebook. This network has 4K nodes but with 88K edges, which is different from Blogs greatly in its high density of connection, and $\frac{\#edges}{\#nodes} = 21.84$.

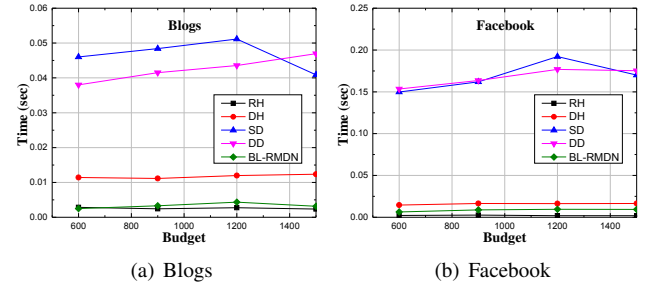


Figure 1. Running time under datasets Blogs and Facebook with $K = 30$

We apply IC model to BRMDN and compare the results with some state-of-the-art heuristic based influence maximization algorithms. We list those algorithms as follows,

RandomHeuristic(RH): It is a baseline for heuristic approaches. It simply selects K random vertices in the graph, which is also evaluated in [2] and [10].

DegreeHeuristic(DH): It intuitively selects K vertices which have the largest degrees in the given graph.

SingleDiscount(SD): A simple degree discount heuristic where each neighbor of a newly selected seed decreases its degree by one, proposed in [10].

DegreeDiscount(DD): Compared to SingleDiscount, it is a refined heuristic method [10].

The greedy algorithms perform very well in terms of influence spread but are intolerable slow for large networks. For example in our experiments, the general greedy algorithm spends 20.78 hours running dataset Blogs with $R = 5$ (normally we have $R = 1000$), while one degree based heuristic algorithm spends about 0.05 seconds, which almost is 1,500,000 times faster. Therefore we do not compare with them in this paper. We gain all the results on a server computer with 24 cores of Intel(R) Xeon(R) CPU E5-2640 2.50GHz and 128G Memory.

For the IC model with relatively large propagation probability p , the influence spread is not very sensitive to different

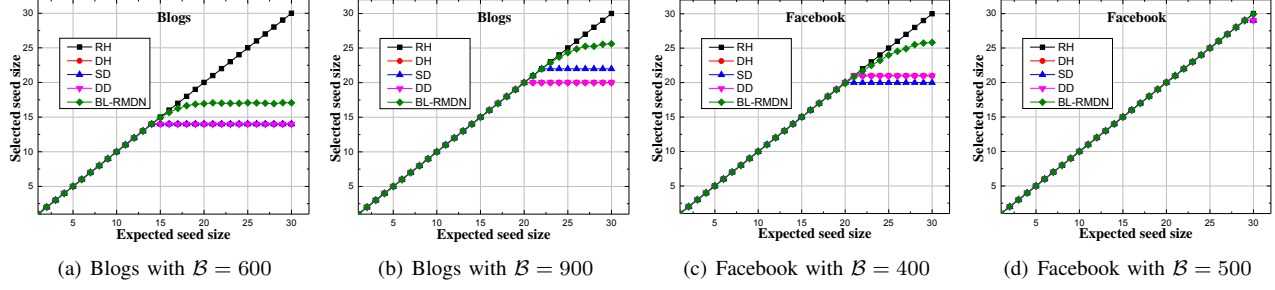


Figure 2. Performance under IC model for dataset Blogs and Facebook with different budget \mathcal{B}

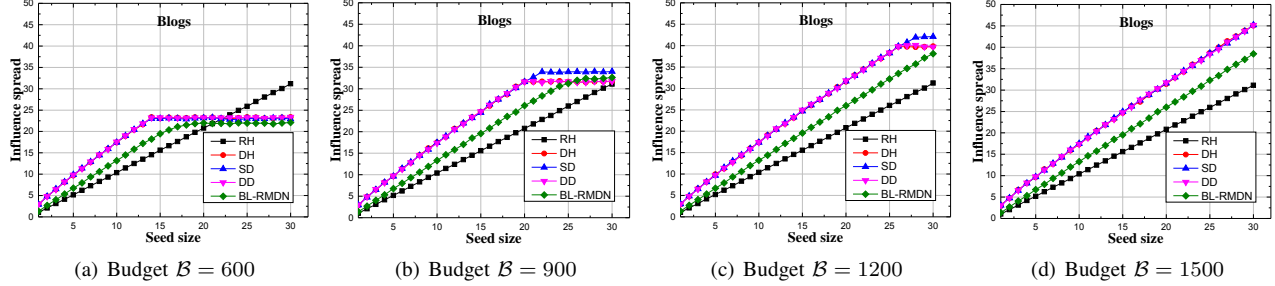


Figure 3. Performance under IC model for dataset Blogs with different budget \mathcal{B}

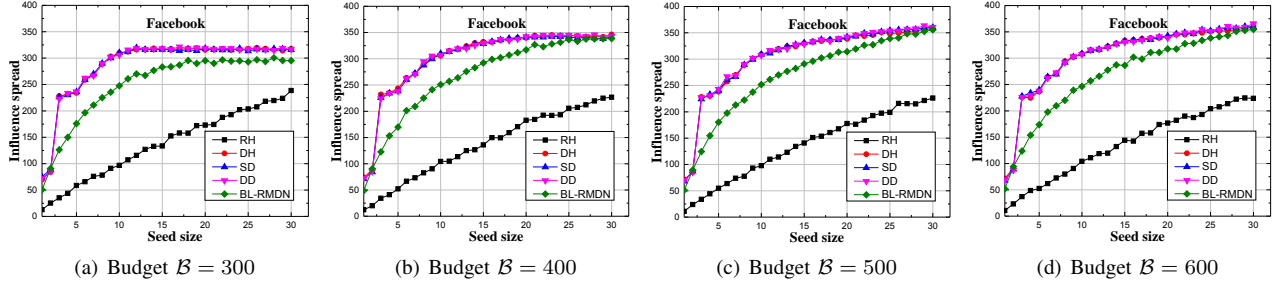


Figure 4. Performance under IC model for dataset Facebook with different budget \mathcal{B}

algorithms. When simulating propagation process with IC model, we set transmitting probability to a relative small number 0.01. To correctly evaluate our method with a reasonable precision, we set our iteration number $R = 1000$. We use PageRank Based Cost Model to evaluate the cost of every node in all the aforementioned algorithms, and we set $\lambda = 100, \delta = 0.5$.

In scale-free networks, the most nodes are sparsely connected, while the hub nodes are densely connected. In the logarithmic coordinate system, a network having scale-free property is a straight-line with gradient valued between -2 and -3. We use the method proposed in [19] [20] to estimate γ , the result is listed in Table II.

B. Experimental Results

The results show that under IC model, BRMDN performs almost as good as DegreeDiscount according their influence spread, while achieves great speedup with respect to running time.

Influence spread: Considering influence spread in Figure 3 and Figure 4, we can see that DegreeDiscount, DegreeHeuristic, SingleDiscount achieve the best; RandomHeuristic, without any surprise, performs the worst; BRMDN approximately approaches to the best algorithms by different ratios with respect to different datasets. In Figure 3, with the increasing of seed set size K , BRMDN gets the result more and more close to DegreeDiscount. Let ∇_{fig} denotes the ratio between the influence spread of BRMDN and DD in Figure fig , then from Figure 3 we have $\nabla_{3(a)} = 95\%$, $\nabla_{3(b)} = 96\%$, $\nabla_{3(c)} = 91\%$, $\nabla_{3(d)} = 85\%$. While in Figure 4, we can see that when the seed set size $K < 15$, BRMDN gets a result approximate to DegreeDiscount with a ratio less than 90%. But when K grows near to 30, BRMDN achieves almost as good as DegreeDiscount. Finally we have $\nabla_{4(a)} = 93\%$, $\nabla_{4(b)} = 99\%$, $\nabla_{4(c)} = 99\%$, $\nabla_{4(d)} = 97\%$. From Figure 3 and Figure 4, BRMDN performs very well in terms of influence spread.

Running time: From Figure 1, we can see that for every

algorithm, the running time is almost a horizontal line which indicates that budget \mathcal{B} has little effect on running time. From Figure 1, we can see that RH runs fastest, second comes BRMDN, DH is the third, SD and DD are the slowest. Specifically, from Figure 1(a) BRMDN is almost 14 times faster than DD, and from Figure 1(b) BRMDN is almost 19 times faster than DD. For datasets Blogs and Facebook, they have relative small number of nodes, so DH's running time is approximate to BRMDN. But from Table I, we can see that when OSN has a large number of nodes (n), BRMDN will show its great speedup.

Noting that in Figure 3(a), when $K > 23$, RH outperforms other algorithms. From Figure 2(a), we find that when budget \mathcal{B} is very limited, other algorithms find a small number of nodes with high cost, while RH can find a large number of nodes with low costs, and finally RH can achieve a better influence spread. From Figure 4, when budget $\mathcal{B} > 400$, all the algorithms can not improve their performance when K is limited to 30. From Figure 2(d), we can see that it is because the nodes selected by all the algorithms are close to 30, and most influential nodes have been already included.

VI. CONCLUSION

Influence maximization is important for activities like products promotion, information transmission, emergency evacuation etc.. It has been extensively studied to find K nodes within the given budget in OSN to achieve maximal spread under constant-cost (i.e. unit-cost) model. In this paper, we firstly propose PRBC to assess nodes' cost according their importance (influence); secondly research budgeted influence maximization under PRBC and propose BRMDN algorithm; finally run extensive experiments to test the performance of BRMDN. The experimental results show: (1) budget \mathcal{B} has little affects on running time when a specific K is given; (2) BRMDN has almost the same influence spread with SD and DD, but achieves great speedup. Therefore, BRMDN balances well between running time and influence spread.

VII. ACKNOWLEDGMENT

This work was supported by National Basic Research Program of China (973 Program) No.2011CB302302, the National High-tech R&D Program of China under Grant No. SS2015AA020102, Tsinghua University Initiative Scientific Research Program.

REFERENCES

- [1] X. He, G. Song, W. Chen, and Q. Jiang, "Influence blocking maximization in social networks under the competitive linear threshold model." in *SDM*. SIAM, 2012, pp. 463–474.
- [2] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 137–146.
- [3] W. Chen, A. Collins, R. Cummings, T. Ke, Z. Liu, D. Rincon, X. Sun, Y. Wang, W. Wei, and Y. Yuan, "Influence maximization in social networks when negative opinions may emerge and propagate." in *SDM*, vol. 11. SIAM, 2011, pp. 379–390.
- [4] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 1029–1038.
- [5] S. Han, F. Zhuang, Q. He, and Z. Shi, "Balanced seed selection for budgeted influence maximization in social networks," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2014, pp. 65–77.
- [6] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, "Complex networks: Structure and dynamics," *Physics Reports-review Section of Physics Letters*, vol. 424, pp. 175–308, 2006.
- [7] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." 1999.
- [8] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.
- [9] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 420–429.
- [10] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 199–208.
- [11] P. Bonacich, "Factoring and weighting approaches to status scores and clique identification," *Journal of Mathematical Sociology*, vol. 2, no. 1, pp. 113–120, 1972.
- [12] H. Nguyen and R. Zheng, "On budgeted influence maximization in social networks," *Selected Areas in Communications, IEEE Journal on*, vol. 31, no. 6, pp. 1084–1094, 2013.
- [13] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 29–42.
- [14] R. Cohen and S. Havlin, "Scale-free networks are ultrasmall," *Physical review letters*, vol. 90, no. 5, p. 058701, 2003.
- [15] L. A. Adamic and B. A. Huberman, "Power-law distribution of the world wide web," *Science*, vol. 287, no. 5461, pp. 2115–2115, 2000.
- [16] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [17] Q. Hu, Y. Gao, P. Ma, Y. Yin, Y. Zhang, and C. Xing, "A new approach to identify influential spreaders in complex networks," in *Web-Age Information Management*. Springer, 2013, pp. 99–104.
- [18] J. Leskovec and J. J. McAuley, "Learning to discover social circles in ego networks," in *Advances in neural information processing systems*, 2012, pp. 539–547.
- [19] A.-L. Barabási, R. Albert, and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A: Statistical Mechanics and its Applications*, vol. 272, no. 1, pp. 173–187, 1999.
- [20] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law distributions in empirical data," *SIAM review*, vol. 51, no. 4, pp. 661–703, 2009.

Using implications from FCA to represent a two mode network data

Sebastião M. Neto, Mark A. J. Song,
Luis E. Zárate

Centro Universitário UNA
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte – MG – Brasil
mark@prof.una.br,
sebastiaoendesneto@gmail.com,
zarate@pucminas.br

Sergio M. Dias

Serviço Federal de Processamento de Dados
SERPRO
Belo Horizonte – MG – Brasil
sergio.dias@serpro.gov.br

Abstract – In a world of ever-growing connectivity, full of connections between people and objects, new multidisciplinary complex network analysis needs to arise. This work presents a solution to analyze an Internet Service Provider database using a formal concept analysis element named implications and complex network techniques. Our goal is to analyze access to the 25 most visited websites to find access patterns. We selected 9 time intervals in one week. Data were converted to a clarified formal context and the FindImplications algorithm was used to extract implications sets. These sets were cross-checked to look for patterns. The implications were used to explore the complex network substructures. As a result, we found access patterns that guarantee that whenever premise websites are accessed, so are conclusion websites. This result can aid in creating security policies and network configurations to help predict future accesses. Without this technique relationships between events nodes (websites) of a two mode network could not be identified.

I. INTRODUCTION

In the last few years, attention has turned towards the increasing complexity of the connected world, as noted by Easley and Kleinberg [1]. This connectivity is propelled by a variety of factors, such as the Internet itself, telephone networks and the speed with which information travels around the globe. These factors enable the genesis of social networks formed by relationships between people. Motivated by the interconnected world, research has surfaced and disciplines interlink to contribute with techniques and new perspectives for the analysis of these complex networks.

Currently, social network analysis is focused on the discovery of social relationship patterns. These relationships can occur between subjects, events, or subjects and events. According to Getoor and Diehi [2], in some cases some rela-

tionships are not observed. Therefore, it is of general interest to unveil hidden substructures as possible and potential communities.

However, the identification of substructures demands work towards perfecting methods to clarify network visualization and extract representations and important knowledge from them [3,4]. We propose a implications-based computational models that allows for the extraction of new knowledge and better visualizations.

The knowledge extraction, in turn, are done via Formal Concept Analysis (FCA) [5], which is a mathematical research field introduced by Rudolf Wille and has found use in different fields. Due to its potential in knowledge representation, FCA can also benefit complex network analysis, as discussed in related work section.

In this work, we used implications seeking to increase our knowledge on complex networks and innovates by using implications to find patterns, build graphs and conduct analyses in a two mode network data.

II. COMPLEX NETWORKS AND LIMITATIONS

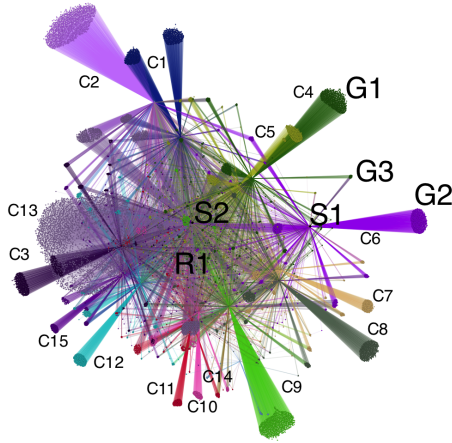
Graph theory [6] is a framework which enables the representation of complex networks in a mathematically accurate way. Formally, a network can be represented by a graph $G = (N, L)$ directed or not, where $N \neq \emptyset$ and L are sets of pairs, sorted or not, of elements from N . The elements in $N \equiv \{n_1, n_2, \dots, n_M\}$ are vertices of graph G . The elements in $L \equiv \{l_1, l_2, \dots, l_K\}$ are the edges.

Networks that contain vertices of a single type are called one mode network data. When networks contain vertices of two or more types, such as vertices that represent users and websites, they are called two mode network data [7].

Fig. 1 shows an example of a network that represents user connections to websites. The resulting graph is a bipar-

(DOI reference number: 10.18293/SEKE2015-085)

Figure 1: Network generated from the accesses of 20.115 users in a day to 15 preselected websites.



tite directional graph, with vertices representing users connected to vertices representing visited websites. It is a two mode network data. A cluster finding algorithm, *Clustering Chinese Whispers* [8], found 15 groupings (C1-C15). These users have a greater probability of communicating and constituting a social network [9][10]. In the groupings G1 and G2, each user accessed a single website. Therefore, they are so next to each other [11]. The grouping G2 accessed the website S1. Users of grouping G3 accessed two websites and, therefore, is positioned between G1 and G2. Website S2 (Google), located in the center of the graph, have a vast number of user connections and its centrality corroborates its importance in the graph. The Region R1, made up of users who accessed many websites, ended up not grouped and becoming dispersed.

Relationships between websites are not easily observed. This begs the question: how to determine which access to a set of websites implies in an access to another website? User-website relationships are perceptible, they are represented by graph edges. However, inter-website relationships cannot be inferred. Freeman and White [12] exposed this limitation in the representation of networks containing nodes of two types (two mode network data) and suggested the concept lattices as the best option. In our work, we showed that implications along with complex network techniques can also help understand this kind of network.

III. FORMAL CONCEPT ANALYSIS AND APRIORI

FCA [5] offers the formalization of a concept, which is made by an intention and an extension. The extension corresponds to all objects belonging to the concept, while the intention represents all attributes shared by the aforementioned objects. FCA allows us to identify object groups with a specific meaning that share common attributes. According to Ganter and Wille [5], FCA revolves around four fundamental elements: formal contexts, formal concepts, concept lattices and implications.

A formal context is usually represented by a table, in which rows represent objects and columns represent attributes. When an object possesses an attribute we have an incidence represented by an “X” [5].

Formally, a formal context has the notation $K := (G, M, I)$, where G is a set of objects (rows), M is a set of attributes (columns) and I are incidences, defined as $I \subseteq G \times M$. If an object $g \in G$ and an attribute $m \in M$ are in relationship I , their representation is $(g, m) \in I$ or gIm , which reads as “object g has attribute m ”.

Given a subset of objects $A \subseteq G$ of a formal context $K := (G, M, I)$, there is an attribute subset of M common to every object of A , even if empty. Likewise, given a set $B \subseteq M$, there is an object subset that shares the attributes of B , even if empty. These relationships are defined by derivation operations [5]: $A' := \{m \in M | gIm \forall g \in A\}$ and $B' := \{g \in G | gIm \forall m \in B\}$

From the formal contexts we obtain formal concepts, defined as pairs (A, B) , where $A \subseteq G$ is called extension and $B \subseteq M$ and is called intention, and they must follow the conditions $A = B'$ and $B = A'$ [5].

With all formal concepts sorted hierarchically by order of inclusion \subseteq we can build the concept lattice.

Implications exist between two attribute subsets of a formal context. Formally, an implication can be expressed as follows. Considering a context $K := (G, M, I)$ satisfying implications $Q \rightarrow R$, $Q, R \subseteq M$, if for every $g \in G$, gIq for every $q \in Q$ implies gIr for every $r \in R$. These implications are normally used in data mining to find dependencies [4].

It is important to note that association rules obtained by the famous APRIORI [13] algorithm are usually expressed with a confidence interval of 0% to 100%. Implications based on existing formal concepts always yield a confidence rate of 100%. Therefore, depending on what the attribute represents, like websites, we can establish an access pattern to related websites.

IV. RELATED WORK

Poelmans et al [14] presents a “semiautomatic” process to expose a network of criminal organizations and their members. They built a lattice of suspected drug dealers and then a lattice containing suspect profiles, that allowed the identification of criminal networks.

Freeman [15] found several important complex network analysis elements through lattice observations, like cliques and bridging cliques, showing that these elements can be observed by FCA and used to facilitate the analysis of complex networks.

Cuvelier and Aupaure [16], analyzed tweets about a specific subject. By means of FCA, the authors were able to establish relationships between messages and, after representing them in the lattice, use data filtering criteria to assemble a topographical network graph to help clarify the information obtained.

Aufaure and Le Grand [17], who describe lattice expressiveness, especially when associated with ontologies, as a benefit of this FCA use. The work is a compilation of case studies. Among the conclusions, they have observed that lattices allow researchers to find deterministically-overlapping groupings, which can be labeled using extensions and intentions.

Our work differs by searching implications for a way to provide more knowledge about the influence of event type vertices in two mode network data networks when looking for substructures.

V. METHODOLOGY

Freeman [12] mentioned the problem of finding the relationships between event type vertices in two mode network data. Here, this problem is attacked by transforming network data into a formal context, extracting implications and finally make a network composed by sets of websites (premises and conclusions) connected by edges (implications). This way, we turn a two mode network data into a one mode network data. With that, we expose existing relationships between event type vertices, which was not feasible in the original network. This would render every complex network technique applicable to the analyzed networks.

The FindImplications [18] algorithm, gather from [19], was used to extract implications. It takes a formal context as input and looks for an implication coverage. It is known for its completeness, being capable to extract all implications.

The object clarification process consists in eliminating duplicated objects. Objects with the same attribute sets can be discarded when obtaining implications, since they do not influence the result [5]. So, we have only considered clarified contexts.

VI. EXPERIMENT AND RESULT ANALYSIS

The database used was provided by a Brazilian cable Internet Service Provider, with anonymous access data. Records refer to accesses made in march 2009, adding up to a total of 6,319,333 accesses.

Targeting the 25 most accesses websites, we found 165,659 (24,9% of the overall access total) accesses made by 29,319 distinct users in the first week of the month. The aiming was to find access patterns among week days. The data were converted to a formal context with websites as attributes and users as objects. Websites from the same domain, for example “www.globo.com”, “ads.globo.com” and “bbb.globo.com” were grouped together for simplification sake, and categorized only as “globo”. The number of attributes was reduced to 15.

Nine formal contexts were generated. Table I shows contexts, the time of day in which accesses were logged, the number of users in the context (objects) and the number of profiles (user sets with identical accesses) in the clarified context.

TABLE I. CONTEXTS AND THEIR CHARACTERISTICS

Context	Access Period	Users	Profiles
K1	Monday from 8 AM-6 PM	11.387	715
K2	Tuesday from 8 AM-6 PM	11.095	710
K3	Wednesday from 8 AM-6 PM	10.829	672
K4	Thursday from 8 AM-6 PM	10.860	700
K5	Friday from 8 AM-6 PM	10.633	683
K6	Friday from 10 PM-2 AM	5.445	315
K7	Saturday from 10 PM-2 AM	5.172	294
K8	Sunday from 10 PM-2 AM	5.482	309
K9	Wednesday from 10 PM-2 AM	5.487	296

The amount of implications extracted and the number of intersections between contexts are shown in Table II.

TABLE II. IMPLICATIONS AND INTERSECTIONS

Cxt	N.R	K2	K3	K4	K5	K6	K7	K8	K9
K1	1.246	50	35	20	51	0	2	0	0
K2	876		29	13	19	2	2	0	0
K3	955			37	60	0	5	0	1
K4	835				46	0	2	0	0
K5	919					1	0	0	1
K6	205						0	0	1
K7	123							0	3
K8	155								0
K9	149								

In addition to the aforementioned intersections, other intersections were found, like between K1, K2, K3 and K4, and the following implication rule, common to all contexts, was also found:

$$\{orkutgstatic, ad.doubleclick.net, yahoo\} \rightarrow \{google\} \quad (1)$$

In rule (1), with 100% certainty, we identified an access pattern which repeated itself in 4 days of the week, from Monday to Thursday. These websites are theoretically harmless. However, in case of a rule which exposes improper behavior, with a premise website set that leads to dangerous conclusion websites, it can be used as a malicious behavior pattern and have alerts and special controls associated to it.

The support for a rule is the percentage of objects that follow the rule relative to the total amount of objects. For rule (1), the approximate support percentage was approximately 0.11% for the non-clarified context K1. Thus, this rule identifies 13 users. For non-clarified context K2, with 11,095 users, the rule has an approximate support rate of 0.08%, identifying 9 users. Context K3, which has 10,829 users, yielded a support rate of 0.06% for the rule, identifying 7 users. For context K4, the support rate was also approximately 0.06%, and the number of identified users was also 7 since K4 has 10,860 users. With this access pattern, we reached a total of 32 distinct identified users, of which four followed the pattern in at least two days.

If we take rule (1) as suspicious user behavior, we could start a detailed control process, tailored to recurring users. As soon as one of them or, if desired, one of the 28 others accesses a premise website, a security policy can be initiated.

The network formed by the implications yields a bipartite, directed graph, with edges going from the premise to the conclusion.

Figure 2: Approximate depiction of the network formed by implications

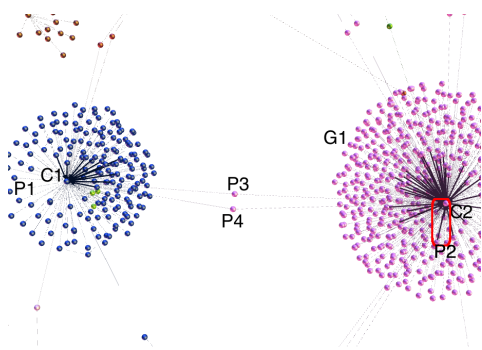


Fig. 2 shows an approximate depiction of a network formed by implications from contexts K3 and K5. Groupings, like G1, indicate premises (websites) with equal conclusions. The algorithm *Clustering Chinese Whispers* identified 449 groupings, implications with the same conclusion. Vertex P1 represents a premise linked to a conclusion C1 (google, yahoo). Vertex P2 is linked to conclusion C2 (google), forming an implication that was recurrent in both days (darker edge in the highlighted area). By making use of strong ties principles [1], it is possible to determine that premises linked to the same conclusion represent users who have similar behavior. If the access to conclusion sites raises concern, like increased bandwidth consumption, when one of the premises occurs, automatic configurations to avoid network bottlenecks can be triggered. Premises P3 and P4 are linked to more than one conclusion, indicating ambiguous behavior for the premises. This only took place because the graph is formed with implications from two days. None of these observations, made from the network composed by premise and conclusion website sets, would have been possible if the original data was visualized and analyzed.

VII. CONCLUSIONS

The main goal of this work was to associate FCA and complex networks to allow us to uncover access patterns (recurrent and ambiguous), in addition to enabling a better representations of relationships between event type elements in two mode network data complex networks.

Implication obtained from user connections made the discovery of access patterns viable. By obtaining rule intersections, patterns and recurring users were identified. The network formed by the rules found ambiguous premises and premise groups with the same conclusion. Examples of use for this type of information were presented, like automatic safeguarding measures to improve service provided and security controls.

We intent to add another dimension, time, to collected data for access prediction analyses based on a profile containing access sequences. For highly dimensional formal contexts, execution times of the algorithms become pro-

hibitive, so we intent to find optimizations for these algorithms using new formal context representations (such as Binary Decision Diagrams and parallelism).

VIII. ACKNOWLEDGMENTS

We thank SERPRO and also FAPEMIG, CNPq and CAPES for their financial support.

REFERENCES

- [1] D. Easley and J. Kleinberg, "Networks, crowds, and markets: Reasoning about a highly connected world", Wiley Online Library, 2012.
- [2] L. Getoor and C. Diehl. "Link mining: a survey", ACM SIGKDD Explorations Newsletter, Vol. 7, Issue 2, pp. 3-12, 2005.
- [3] L. C. Freeman, "Visualizing social networks", Journal of social structure, vol. 1 2000.
- [4] L. C. Freeman, "Graphical techniques for exploring social network data", Models and Methods in Social Network Analysis, 2005.
- [5] B. Ganter, G. Stumme, and R. Wille, "Formal concept analysis: foundations and applications", Dresden, Alemanha, Springer, v.3626. 2005.
- [6] B. Bollobas, "Random graphs", Academic Press, London, 1985.
- [7] S. Wasserman and K. Faust, "Social network analysis: methods and applications", New York. Academic Press. 1993
- [8] C. Biemann, "Chinese whispers: an efficient graph clustering algorithm and its application to natural language processing problems", In Proceedings of the first workshop on graph based methods for natural language processing, p. 73-80, Association for Computational Linguistics, 2006.
- [9] L. B. Ronald, "The duality of persons and groups", Social Forces, vol. 53, pp. 181-190, 1974.
- [10] S. L. Feld. "The focused organization of social ties". American Journal of Sociology, 86(5), pp. 1015-1035, 1981.
- [11] M. Jacomy, S. Heymann, T. Venturini, and M. Bastian, "Forceatlas2, a graph layout algorithm for handy network visualization", Paris <http://www.medialab.sciences-po.fr/fr/publications-fr>, 2009.
- [12] L. C. Freeman and D. R. White, "Using galois lattices to represent network data". Sociological methodology, v. 23, pp. 127-146. 1993.
- [13] R. Agrawal, S. Ramakrishnan, "Fast algorithms for mining association rules", In Proc. 20th int. conf. very large data bases, VLDB, vol. 1215, pp. 487-499, 1994.
- [14] J. Poelmans, P. Elzinga, S. Viaene, G. Dedene, and S. Kuznetsov, "A concept discovery approach for fighting human trafficking and forced prostitution", 19th International conference on conceptual structures, lecture notes in computer science, vol. 6828, pp. 201-214, Derby, England: Springer, 2011.
- [15] L. C. Freeman, "Cliques, galois lattices, and the structure of human social groups", Elsevier, Social Networks, 18, pp. 173-187, 1996.
- [16] E. Cuvelier and M. Aufaure, "A buzz and e-reputation monitoring tool for twitter based on galois lattices", In Conceptual Structures for Discovering Knowledge, pp. 91-103, Springer Berlin Heidelberg, 2011.
- [17] M. Aufaure and B. Le Grand, "Advances in FCA-based Applications for Social Networks Analysis", International Journal of Conceptual Structures and Smart Applications (IJCSSA) 1, vol. 1, pp. 73-89, 2013.
- [18] C. Carpineto and G. Romano, "Concept data analysis: theory and applications", Wiley, 2004.
- [19] S. M. Dias and N. J. Viera, "A framework for the development of formal concept analysis algorithms" in portuguese "Um arcabouço para desenvolvimento de algoritmos da análise formal de conceitos", Revista de Informática Teórica e Aplicada, vol. 18, no. 1, p. 31-57, 2011.

How do developers use C++ libraries? An empirical study

Di Wu

Lin Chen *

Yuming Zhou

Baowen Xu

State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, China

nju.wudi@gmail.com

lchen@nju.edu.cn

zhouyuming@nju.edu.cn

bwxu@nju.edu.cn

Abstract—C++ libraries provide an abundance of reusable components for writing high-quality programs and are thus widely adopted by software developers. However, to date there is little work investigating how these libraries are actually used in real software. In this paper, we perform an empirical study to investigate the adoption of C++ standard libraries in open-source applications, with the goal to provide actionable information for developers to help them employ libraries more efficiently. To this end, we analyze 379 historical revisions of 30 applications, containing 149 million lines of C++ code, to conduct the experiment. The experimental results show that: (1) three standard libraries (i.e. Containers Library, Utilities Library, and Strings Library) are significantly more often used than other libraries; (2) the new libraries of C++11 (i.e. Regular Expressions Library, Atomic Operations Library, and Thread Support Library) are significantly less often used than the formerly-established libraries; (3) the deprecated library constructs (i.e. auto pointers, function objects, and array I/O operations) are not used at a declining frequency; and (4) applications with a larger size do not adopt libraries more frequently. Based on these results, we propose four suggestions, which could help developers learn and use C++ libraries in an efficient way.

Keywords- Programming Language, C++, Library, Empirical Study

I. INTRODUCTION

C++ libraries are pervasively used in software development, as they enable developers to write high-quality programs by employing reusable components rather than implementing all code from scratch [4]. To date, various libraries have been provided to help solve problems of different domains. Among these libraries, the Standard C++ Library is the most renowned, since it provides a large set of standardized components that are shipped with identical behavior by every C++ implementation [5]. According to C++11 [3], the latest¹ C++ specification, the Standard C++ Library is constituted by 11 sub-libraries, including 3 new libraries introduced in C++11 and 8 old libraries established in C++98 [1] and C++03 [2]. For brevity, people generally call these sub-libraries as “standard libraries”.

In recent years, many researchers have been devoting to improve the performance of standard libraries. However, few studies focus on how these libraries are actually adopted in real software. This lack of knowledge may bring troubles to software developers, since they do not know which standard libraries are the most commonly used and need their attention to be paid on, whether they have made full use of new

standard libraries, and whether deprecated library constructs are less frequently used.

In this paper, we perform an empirical study to investigate the adoption of C++ standard libraries in open-source applications, with the goal to provide actionable information for developers to help them use libraries more efficiently. To be specific, we propose the following four research questions: (1) RQ1: Which libraries are the most often used? (2) RQ2: Are the new libraries of C++11 used as often as the formerly-established libraries? (3) RQ3: Are the deprecated library constructs used at a declining frequency after C++11 was published? and (4) RQ4: Do applications with a larger size adopt libraries more frequently? The purpose of RQ1 investigates whether there exist a few libraries that are more often used than others. If the most commonly used libraries are found, we may suggest developers, especially the new comers of open-source projects, to focus on understanding and using these libraries. The purpose of RQ2 investigates whether the new libraries of C++11 have been widely used. If the answer is “Yes”, we will have empirical evidence to support that the new features of C++11 have been widely adopted in developing real software. Otherwise, we may advise developers to pay special attention on using applicable new library constructs instead of writing their own code of similar functionality. The purpose of RQ3 investigates whether the deprecated library constructs are gradually infrequently used. Due that auto pointers, function objects, and array I/O operations can be replaced by other advanced features, they have been deprecated since C++11. By investigating RQ3, we can understand whether developers have realized to reduce using these outdated library constructs. The purpose of RQ4 investigates the correlation between system size and the frequency of library use. In previous studies on Java and C# libraries [9, 10], researchers found that applications with different sizes adopt libraries differently. The empirical result for RQ4 can be used to answer whether this conclusion is also applicable to the use of C++ libraries.

In order to answer these research questions, we analyze 379 historical revisions of 30 applications, containing 149 million lines of C++ code, to conduct the experiment. The experimental results show that: (1) three standard libraries (i.e. Containers Library, Utilities Library, and Strings Library) are significantly more often used than other libraries; (2) the new libraries of C++11 (i.e. Regular Expressions Library, Atomic Operations Library, and Thread Support Library) are significantly less often used than the formerly-established libraries; (3) the deprecated library constructs (i.e. auto pointers, function objects, and array I/O operations) are not used at a declining frequency; and (4) applications with a larger size do not adopt libraries more frequently. Based on these results, we propose four suggestions, which could help developers learn and use C++ libraries in an efficient way.

* Corresponding author: Lin Chen; Email: lchen@nju.edu.cn

¹ C++14 was recently approved, but its official specification has not been released. Thus, we still serve C++11 as the latest standard of C++ in this paper.

The rest of the paper is organized as follows. Section II introduces the C++ Standard Library. Section III describes the studied applications, data collection procedure, and data analysis methods. Section IV reports the experimental results, the implications, and the threats to validity of our study. Section V discusses related work. Section VI concludes the paper and outlines the direction for future work.

II. AN OVERVIEW OF THE C++ STANDARD LIBRARY

The C++ Standard Library is a general name for the standardized built-in classes, functions, and macros in C++. The whole standard library is constituted by 11 sub-libraries, which are generally called “standard libraries”. Before C++11, 8 elementary standard libraries were supported. To differentiate them from new libraries of C++11, we call these libraries as “**formerly-established libraries**”. These libraries basically consist of Containers, Iterators, Algorithms, Utilities, Strings, Numerics, Input/Output, and Localizations. The first three libraries together with function objects in the Utilities library constitute STL (the Standard Template Library), which provides generic classes and functions to create and operate common data structures like vectors, queues, and stacks. The other five libraries are specific to language support (as well as general-purpose utilities support), string processing, scientific computation, I/O management, and internationalization support, respectively. Since C++11, three **new libraries** have been introduced. They are Regular Expressions Library, Atomic Operations Library, and Thread Support Library. The first new library is used to perform pattern matching for strings. The other two new libraries are specific to concurrent programming, equipped with low-level (atomics-based) and high-level (thread and task-based) concurrency facilities, respectively. Moreover, three formerly-established library constructs (i.e. auto pointers, function objects, and array I/O operations) are deprecated in C++11. They are no longer supported either due to the low efficiency or due to the advanced replacers.

III. EXPERIMENTAL SETUP

In this section, we first introduce the open-source applications used for investigating our research questions. Then, we report the data collection procedure. Finally, we describe the data analysis methods.

A. Studied Applications

To investigate the proposed research questions, we analyze 30 open-source applications, whose source code is obtained by using *svn* and *git clone* tools. These applications are selected for the following reasons: (1) they cover different application domains listed on <http://sourceforge.net>, thus making the empirical results not skewed to a specific kind of applications; (2) they have a big difference in code size, thus making the result for RQ4 sufficiently reliable; and (3) they are developed as ongoing projects, thus making the experimental data up-to-date. The detailed information of the 30 applications is shown in Table I. As we can see from Table I, these applications cover 10 software domains. Moreover, they vary in age (2 to 16 years) and code size (9 to 4731 KSLOC). For these applications, we use their latest revisions by the end of 2014 to

investigate RQ1, RQ2, and RQ4 and use their historical revisions to investigate RQ3. In our experiment, the historical revisions are regularly selected as the last revisions in each season after September 2011, the release time of C++11. We do not investigate all historical revisions because the code repositories contain many dump revisions, which may pose a threat to the accuracy of our experimental data. For some applications (i.e. PN, SwiftSearch, HTEditor, and ConEmu), only a few revisions are studied. This is either due to their late establishing time or due to the long time intervals between adjacent revisions.

TABLE I. OPEN-SOURCE APPLICATIONS IN THE STUDY

Project	Age	C++ KSLOC of latest revision	# Studied revisions	Total C++ KSLOC ¹	Category
VLC	16	135.825	14	1855.855	Audio & Video
LameXP	5	21.449	14	315.346	
MPC-HC	9	521.267	14	9581.495	Business & Enterprise
MuPDF	11	16.137	14	140.816	
Qucs	12	125.600	13	2323.950	Enterprise
LibreOffice	5	4730.718	14	68554.599	
LeechCraft	8	325.526	14	3819.669	Communications
MirandaNG	3	1087.472	12	11155.677	
KopeteIMClient	13	348.629	14	4009.079	Development
TortoiseGit	7	457.517	14	5153.035	
PN	13	155.059	7	1084.903	Development
KDevelop	16	108.759	14	1388.152	
Warzone2100	10	186.721	12	2233.427	Games
Pentobi	4	30.818	14	375.543	
SuperTuxKart	8	369.355	14	3476.652	Graphics
Blender	13	600.906	14	6852.224	
LuminanceHDR	13	38.828	13	452.887	Graphics
FreeCAD	4	1185.593	14	15992.063	
GoldenDict	6	75.008	14	748.038	Home & Education
Kiwix	8	63.863	14	1060.158	
SUMO	13	132.400	14	1651.322	Home & Education
rr	4	18.245	14	66.944	
Trimph4php	3	80.211	10	562.450	Science & Engineering
RStudio	2	127.943	14	1300.345	
KmyMoney	3	146.252	14	1993.554	Security & Utilities
SwiftSearch	3	8.556	6	43.196	
HTEditor	13	95.517	7	712.315	Utilities
ConsoleZ	8	65.056	14	840.411	
NVDA	9	11.291	14	139.735	System Administration
ConEmu	2	203.471	5	955.516	

B. Data Collection

We collect the experimental data by using “Understand” [17], a tool that automatically analyzes the source code of applications without manual configuration. To be specific, the data is collected by the following steps. At the first step, we obtain C++ files by using the “C++ Strict” option provided by “Understand” and build an Understand database for each studied application. At the second step, we process Understand databases to identify the use sites of standard library constructs, including library classes, library functions, and library macros. Since all standard library constructs are marked with the “std:” namespace, they can be easily detected by running a Perl script which invokes Understand APIs. At

¹ “KSLOC” means “thousand source lines of code (excluding comments)”. Generally speaking, it is equal to “KLOC” (“thousand lines of code”).

the third step, we compare the names of practically used constructs with the names of actual standard library constructs. We do this in order to filter out those fake standard library constructs used by developers. At the fourth step, we divide all examined standard library constructs into the new library group and the formerly-established library group. At the final step, we calculate the KSLOC value and the number of C++ files for each application by looking up the metrics reported by Understand. With these five steps, we can obtain the experimental data set, which consists of: (1) the number of use for each standard library (both formerly-established and new libraries); (2) the number of use for deprecated library constructs; (3) the number of use for standard libraries in each application and in its historical revisions; and (4) the KSLOC value and the number of files in each application.

C. Data Analysis

In order to answer RQ1, RQ2, and RQ3, we apply the Wilcoxon signed-rank test to examine whether two groups of data have a significant difference. More specifically, for RQ1, we compare the percentages of use for the 11 standard libraries in pair-wise. Here, the percentage is calculated as the number of use for a specific library divided by the total number of use for all libraries. If a few libraries exceed other libraries in the percentage of use at the significance level of 0.05, we will accept them as the most commonly used standard libraries. Otherwise, we will conclude that there is not an outstanding library that is more often used than others. For RQ2, we compare the percentage of use for each new library with the percentage of use for each formerly-established library. If new libraries show a significant difference (significance level = 0.05) from the formerly-established libraries in the percentage of use, we will conclude that the new libraries and formerly-established libraries are not equally commonly used. Otherwise, we will fail to reject the hypothesis that “new libraries are as often used as formerly-established libraries”. For RQ3, we compare the densities of use for the deprecated library constructs in each season after C++11 was published. The densities are calculated both at line level (number of use for deprecated library constructs per KSLOC) and at file level (number of use for deprecated library constructs per file). Here, we use the density instead of the raw number of library construct use in order to avoid the impact brought by the change of system size. The answer to RQ3 will be “Yes” if the density value in one season (for instance, Dec. 2014) is significantly lower than the density value in the former season (for instance, Sep. 2014). Otherwise, we will fail to conclude that the deprecated library constructs are used at a declining frequency after C++11 was officially released. After performing each Wilcoxon signed-rank test, we further apply the Cliff’s δ , which is used for median comparison, to examine whether the magnitude of difference is important from the viewpoint of practical application [6]. By convention, the magnitude of the difference is either trivial ($|\delta| < 0.147$), small (0.147-0.33), medium (0.33-0.474), or large (> 0.474) [7].

In order to answer RQ4, we use the Spearman’s rank correlation analysis to examine whether the size of applications is significantly positively correlated to the frequency of library use. In previous studies [9, 10], researchers found that

applications with different sizes adopt libraries differently. More specifically, larger applications tend to have more library uses. However, the raw number of library use cannot effectively reflect the frequency of library use in different applications, because larger applications usually have more functionalities and not surprisingly have more library uses. In order to remove the impact of different system size, here we use the density to replace the raw number of library use. More specifically, we first calculate the density of library use for each application. The densities are calculated both at line level (number of library use per KSLOC) and at file level (number of library use per file). Then, we calculate the Spearman’s coefficient (ρ) of the correlation. In particular, the p-value is employed to examine whether the correlation is significant at the significance level of 0.05. If the calculated p-value is less than 0.05, we will conclude that applications with a larger size adopt libraries more frequently. Otherwise, we will have a conclusion that the size of application does not significantly positively correlates to the frequency of library use.

IV. RESULTS AND IMPLICATIONS

In this section, we report in detail the experimental results and discuss their implications.

A. RQ1: Which libraries are the most often used?

We employ the result from the Wilcoxon signed-rank analysis for the percentage of library use to answer RQ1. In particular, we apply Figure I to describe the percentage of use for each library. In this figure, each boxplot shows the median (the horizontal line within the box), the 25th and 75th percentiles (the lower and upper sides of the box), and the mean value (the small red rectangle inside the box). By observing Figure I, we can see that the percentages of use for three libraries (i.e. Containers, Utilities, and Strings) are obviously larger than the percentages of use for other libraries (i.e. Iterators, Algorithms, Numerics, I/O, Localizations, Regular Expressions, Atomic Operations, and Thread Support), indicating that these three standard libraries are the most commonly used by developers. The data listed in Table II confirms our observation from Figure I. This table displays the result from the Wilcoxon signed-rank analysis for the pair-wise comparisons between three standard libraries (the first row) and the other eight standard libraries (the first column). In particular, we report the significance (p-value) and the magnitude (Cliff’s δ) of the difference, respectively. To be specific, for the Containers Library, it significantly outperforms other eight libraries in the percentage of use (all p-values < 0.001). Moreover, the effect sizes are large in terms of Cliff’s δ ($0.804 \leq |\delta| \leq 0.966$). The Utilities Library, as expected, shows a similar result, and the effect sizes are considerably large ($0.931 \leq |\delta| \leq 0.973$). For the Strings Library, its percentage of use is also significantly larger than the other eight libraries, with seven p-values less than 0.001 and one p-value equaling to 0.017. Moreover, the effect sizes are either small ($\delta = 0.329$), moderate ($\delta = 0.393$), or large ($0.482 \leq |\delta| \leq 0.862$). To summarize, the core observation from Table II is that three new libraries significantly outperform the other eight libraries in the percentage of use and the magnitude of difference is relatively large. Therefore, we have the

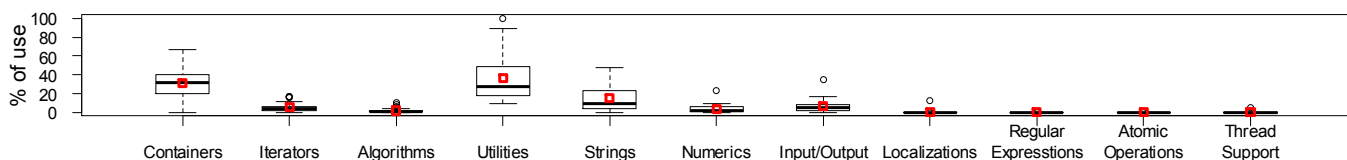


Figure I. Boxplot showing the percentage of use for standard libraries

TABLE II. RESULTS OF WILCOXON SIGNED-RANK ANALYSIS FOR RQ1

	Containers		Utilities		Strings	
	p	δ	p	δ	p	δ
Iterators	<0.001	0.820	<0.001	0.949	<0.001	0.393
Algorithms	<0.001	0.911	<0.001	0.966	<0.001	0.642
Numerics	<0.001	0.859	<0.001	0.973	<0.001	0.482
I/O	<0.001	0.804	<0.001	0.931	0.017	0.329
Localizations	<0.001	0.947	<0.001	0.963	<0.001	0.784
Regular exp.	<0.001	0.963	<0.001	0.967	<0.001	0.853
Atomic op.	<0.001	0.966	<0.001	0.967	<0.001	0.862
Thread sup.	<0.001	0.959	<0.001	0.967	<0.001	0.838

* All the p-values are BH-adjusted

following conclusion for RQ1: **three standard libraries (i.e. Containers Library, Utilities Library, and Strings Library) are significantly more often used than the other libraries.**

In order to find out which library constructs play a key role in Containers, Utilities, and Strings, we further pick out the most commonly used library constructs on ground of their number of use. All library constructs are divided into three groups, namely library classes, library functions, and library macros. According to the obtained result, library functions (73.95%) are more often used than library classes (7.89%) and library macros (18.16%). One possible explanation for this is that library functions are generally used as APIs and they are widely applied to operate elementary data structures (for instance, bitsets, shared pointers, maps, etc). Also, we find that many library classes are implemented as templates, especially the STL templates (for instance, map, set, list, and vector) and the Utilities templates (for instance, tuple, pair, bitset, numeric_limits, shared_ptr, and auto_ptr). This indicates that library templates play an important role in creating the basic data structures, which is in line with our previous findings about the utilization of templates [16]. For library macros, we find that the most commonly-used macros are inclusive members of Utilities. This result is not surprising, because an important role of the Utilities Library is to provide language support with built-in macros like UINT8_MAX, INT16_MAX, EXIT_SUCCESS, etc.

Implication. From the empirical results for RQ1, we advise developers, especially the new comers of open-source projects, to be proficient with the usage of Containers, Utilities, and Strings. Since these standard libraries are the most often used in real software development, adopting them effectively is beneficial to increase the efficiency of programming.

B. *RQ2: Are the new libraries of C++11 used as often as the formerly-established libraries?*

We employ the result from the Wilcoxon signed-rank analysis for the percentage of new library use to answer RQ2. Here, we exclude the experimental data provided by the applications which were established before C++11 was

released. We do this mainly because these applications have already existed before the delivery of new libraries, thus investigating their use of new libraries may pose a threat to the result for RQ2. To eliminate this negative impact, we only employ the data of new library use in the applications which were established after the delivery of C++11. Table III shows the results for the pair-wise comparisons between the adoption of new libraries (the first row) and the adoption of formerly-established libraries (the first column). In particular, we report the significance (p-value) and the magnitude (Cliff's δ) of the difference, respectively. To be specific, for the Regular Expressions Library, its percentage of use is significantly different from the percentage of use for seven formerly-established libraries (p-values ≤ 0.016). Moreover, the effect sizes are large in terms of Cliff's δ ($0.877 \leq |\delta| \leq 1$). The only exception is the Localizations Library, which does not show a significant difference from Regular Expressions (p-value = 0.281). For the other two new libraries (i.e. Atomic Operations and Thread Support), they show a similar result as the Regular Expressions Library. From this reasoning, we conclude that new libraries and formerly-established libraries are differently used. Actually, **the new libraries of C++11 are much less often used than the formerly-established libraries.**

TABLE III. RESULTS OF WILCOXON SIGNED-RANK ANALYSIS FOR RQ2

	Regular exp.		Atomic op.		Thread sup.	
	p	δ	p	δ	p	δ
Containers	0.010	-1.000	0.010	-1.000	0.010	-1.000
Iterators	0.016	-0.877	0.016	-0.889	0.034	-0.827
Algorithms	0.016	-0.877	0.016	-0.889	0.019	-0.802
Utilities	0.010	-1.000	0.010	-1.000	0.010	-1.000
Strings	0.016	-0.877	0.016	-0.889	0.019	-0.877
Numerics	0.010	-1.000	0.010	-1.000	0.010	-0.926
I/O	0.016	-0.877	0.016	-0.889	0.019	-0.877
Localizations	0.281	-0.333	0.100	-0.444	0.419	-0.309

* All p-values are BH-adjusted; p-values > 0.05 are shown in grey background.

Implication. The result for RQ2 is opposed to our initial expectation that new libraries and formerly-established libraries should be equally used. One possible explanation for this is that most developers are still not familiar with the usage of new libraries, as the new libraries are been a part of the C++ standard for only three years. For this reason, we highly recommend developers to pay special attention on learning the usage of new libraries (i.e. Regular Expressions Library, Atomic Operations Library, and Thread Support Library) and employ them when they need to write string matching or concurrent programs.

C. *RQ3: Are the deprecated library constructs used at a declining frequency after C++11 was published?*

We employ the result from the Wilcoxon signed-rank analysis for the density of use for deprecated library constructs

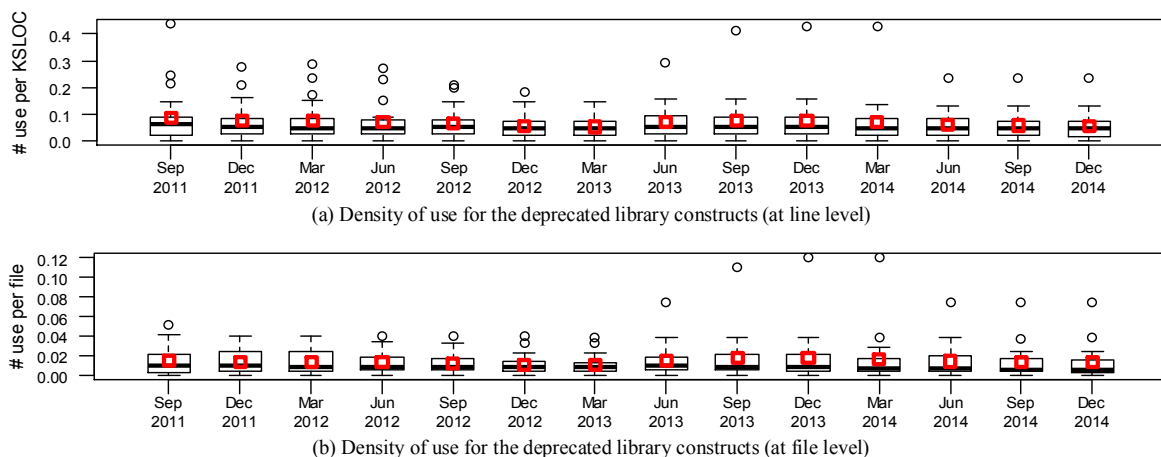


Figure II. Density of use for the deprecated library constructs

to answer RQ3. In particular, we use Figure II to describe the density values both at line level (number of use per KSLOC) and at file level (number of use per file). In Figure II, each boxplot shows the median (the horizontal line within the box), the 25th and 75th percentiles (the lower and upper sides of the box), and the mean value (the small red rectangle inside the box). By observing the two subfigures, we do not see an obvious declining trend for the density values from Sep. 2011 to Dec. 2014, indicating that the deprecated library constructs are not decreasingly frequently used after C++11 was released. The data listed in Table IV confirms our observation from Figure II. In Table IV, we show the Wilcoxon signed-rank analysis results for comparing two adjacent seasons since September 2011. Of the 13 comparison results listed in the “Line level” group, we totally find 6 significant results (p -values < 0.05), whose effect sizes are either trivial or small in terms of Cliff’s δ ($0.008 \leq |\delta| \leq 0.163$). By observing the “File level” column, however, we only have 3 significant results, whose effect sizes are relatively negligible ($0.044 \leq |\delta| \leq 0.108$). To summarize, the core observation from Table IV is that the density of use for the deprecated library constructs does not significantly decrease from late 2011 to the end of 2014. From this reasoning, we draw the conclusion for RQ3 as **the deprecated library constructs are not used at a declining frequency after C++11 was published.**

TABLE IV. RESULTS OF WILCOXON SIGNED-RANK ANALYSIS FOR RQ3

Groups for comparison	Line level		File level	
	p	δ	p	δ
Dec.2011 vs. Sep.2011	0.060	-0.006	0.133	-0.008
Mar.2012 vs. Dec.2011	0.358	-0.039	0.529	0.003
Jun.2012 vs. Mar.2012	0.032	-0.025	0.087	0.008
Sep.2012 vs. Jun.2012	0.060	-0.017	0.116	-0.019
Dec.2012 vs. Sep.2012	0.005	-0.163	0.007	-0.108
Mar.2013 vs. Dec.2012	0.032	-0.047	0.031	-0.044
Jun.2013 vs. Mar.2013	0.157	0.015	0.446	0.119
Sep.2013 vs. Jun.2013	0.377	0.019	0.534	0.014
Dec.2013 vs. Sep.2013	0.083	-0.055	0.345	-0.033
Mar.2014 vs. Dec.2013	0.074	-0.080	0.095	-0.069
Jun.2014 vs. Mar.2014	0.039	0.008	0.097	-0.003
Sep.2014 vs. Jun.2014	0.032	-0.080	0.031	-0.080
Dec.2014 vs. Sep.2014	0.013	-0.050	0.087	-0.025

* All p-values are BH-adjusted; p-values > 0.05 are shown in grey background.

Implication. One possible explanation for RQ3 is that most developers do not realize that several long-lived library constructs (i.e. auto pointers, function objects, and array I/O operations) have been deprecated since C++11. For this reason, we advise developers to keep an eye on the changes in the new C++ standards and update their code accordingly. In particular, we wish developers to remove the uses of the deprecated library constructs, because these constructs will completely stop to be supported since C++17 [18], the next major revision of the C++ programming language.

D. RQ4: Do applications with a larger size adopt libraries more frequently?

In order to answer RQ4, we use the Spearman’s rank correlation analysis described in Section III.C to examine the correlation between the size of application and the density of library use. Here, we calculate application size both as KSLOC (line-level size) and as the number of files (file-level size), with the purpose to investigate RQ4 from different perspectives and obtain a consistent result. To be specific, the result of Spearman’s rank correlation analysis at line level shows that application’s KSLOC does not significantly correlate to the density of library use (number of library use per KSLOC) (p -value = 0.896). A similar result is reported by the Spearman’s rank correlation analysis at file level, which shows that the number of files and the density of library use (number of library use per file) are not significantly correlated (p -value = 0.799). From this reasoning, we conclude that the size of application is not significantly correlated to the density of library use. In other words, **applications with a larger size do not adopt libraries more frequently.**

Implication. According to Robillard and DeLine [8], library users can efficiently understand an API if they are provided with examples to demonstrate “best practices” for using the API. Thus, it would be valuable work to explore real examples of library use in open-source applications. Since the conclusion for RQ4 indicates that applications of different size do not adopt libraries at different frequency, we suggest newcomers of open-source projects to learn API usage examples by reading the source code of small applications. This can help them obtain better learning effect by avoiding understanding the complex source code of large applications.

E. Threats to Validity

There are four possible threats to validity in this study. The threat to the construct validity is the correctness of library use sites reported by “Understand”. Since many studies have produced reliable empirical results by using “Understand” [17], the data in our study can also be considered as acceptable. The threat to the internal validity is that we do not exclude new library constructs from the formerly-established libraries. But according to our empirical data, the new library constructs only account for a relatively small proportion of the use (1.23%) for formerly-established library use. For this reason, our empirical results are still reliable. The first threat to the external validity is that we only use open-source applications to conduct the experiment. The empirical results may not be applicable to industrial applications, as different ways of software development probably make a difference in the adoption of libraries. The second threat to the external validity is that we only investigate standard libraries. The third-party libraries are not included mainly because they are generally considered not as widely used as standard libraries.

V. RELATED WORK

Due to page limitation, here we only discuss a few studies most related to our work. In recent years, more and more researchers have started to investigate the adoption of software libraries in an empirical way. Torres et al. [9] were among the first to study the usage of Java concurrency libraries and they found a list of commonly-used concurrency library constructs. Also, they concluded that medium to large-sized applications tend to use more concurrency constructs. However, this conclusion was drawn by simply comparing the raw number of library use among small applications (1-20KLOC), medium applications (20-100KLOC), and large applications (>100KLOC). By comparison, we use the Spearman’s rank correlation analysis method to test the relationship between the size of application and the frequency of library use, which can produce a more reliable result. Another related study was an empirical investigation on C# parallel libraries performed by Okur and Dig [10], who showed that applications with different sizes have different adoption trends. However, they only compare the raw number of library use among different applications instead of investigating the frequency of library use. For this reason, this finding is limited to some extent. Before this study, we have already performed an empirical investigation on the adoption of C++ templates [16], which showed that STL predominates the overall use of library templates. Compared with our previous work, this paper investigates the adoption of C++ libraries at a higher level by focusing on the whole C++ Standard Library, not limited to library templates. The other related work includes the investigation on MPI open-source applications [11], the study on Java library use trend [12], the research on Java API popularity [13], the assessment on third-party libraries [14], and the exploration on third-party component reuse [15].

VI. CONCLUSION AND FUTURE WORK

In this paper, we conduct a study on the adoption of C++ libraries in real applications. The whole study is performed by investigating four research questions regarding the most often

used libraries, the difference between the use of the new libraries and the use of the formerly-established libraries, the trend of adopting deprecated library constructs, and the relationship between the size of application and the frequency of library use. By employing inferential statistics, we get reasonable results for the proposed research questions. Based on the empirical results, we give four actionable suggestions, which could help developers, especially the new comers of open-source projects, learn and use libraries efficiently. In the future work, we will investigate more research questions and perform an empirical study on more applications to understand the adoption of C++ libraries in depth.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (61170071, 61432001, 91418202, 61472175, 61472178), the National Natural Science Foundation of Jiangsu Province (BK20130014), and the program B for Outstanding PhD candidate of Nanjing University.

REFERENCES

- [1] ISO/IEC. Information Technology—Programming Languages—C++. ISO/IEC 14882-1998. 1998.
- [2] ISO/IEC. Information Technology—Programming Languages—C++, Second Edition. ISO/IEC 14882-2003. 2003.
- [3] ISO/IEC. Information Technology—Programming Languages—C++, Third Edition. ISO/IEC 14882-2011. 2011.
- [4] N. Josuttis. The C++ Standard Library: A Tutorial and Reference - Second Edition. Addison-Wesley, 2012.
- [5] B. Stroustrup. The C++ programming language - Fourth Edition. Addison-Wesley, 2013.
- [6] E. Arisholm, L. Briand, B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1), 2010: 2-17.
- [7] J. Romano, J. Kromrey, J. Coraggio, J. Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen’s *d* for evaluating group differences on the NSSE and other surveys? In: *Annual Meeting of the Florida Association of Institutional Research*, 2006: 1-3.
- [8] M. P. Robillard, R. DeLine. A field study of API learning obstacles. *Emp. Soft. Eng.*, 16(6), 2011: 703-732.
- [9] W. Torres, G. Pinto, B. Fernandes, J. P. Oliveira, F. Ximenes, F. Castor. Are Java programmers transitioning to multicore? A large scale study of java FLOSS. *SPLASH*, 2011: 123-128.
- [10] S. Okur, D. Dig. How do developers use parallel libraries? *FSE*, 2012: Article No. 54.
- [11] C. Marinescu. An empirical investigation on MPI open source applications. *EASE*, 2014: Article No. 20.
- [12] Y. M. Mileva, V. Dallmeier, M. Burger, A. Zeller. Mining trends of library usage. *IWPSE-Evol*, 2009: 57-62.
- [13] Y. M. Mileva, V. Dallmeier, A. Zeller. Mining API popularity. *TAIC PART*, 2010: 173-180.
- [14] S. Blom, J. Kiniry, M. Huisman. A structured approach to assess third-party library usage. *ICECCS*, 2013: 212-221.
- [15] W. Schwitek, S. Eicker. A study on third party component reuse in Java enterprise open source software. *CBSE*, 2013: 75-80.
- [16] D. Wu, L. Chen, Y. Zhou, B. Xu. An empirical study on the adoption of C++ templates: library templates versus user defined templates. *SEKE*, 2014: 144-149.
- [17] SciTools Understand. <https://scitools.com/>.
- [18] C++17. <http://en.wikipedia.org/wiki/C%2B%2B17>.

A Case Study Approach: Iterative Prototyping Model Based Detection of Macular Edema in Retinal OCT Images

Sadaf Sahar, Sadaf Ayaz, M.Usman Akram, Dr. Imran Basit(AFIO)

Department of Computer Engineering
College of Electrical & Mechanical Engineering NUST
Rawalpindi, Pakistan

sadafsahar21@gmail.com, sadafayaz32@gmail.com, usmakram@gmail.com, drimranbasit@gmail.com

Abstract—Highly Reliable Automated medical diagnosis systems are of critical importance. Such systems aid in early detection of diseases and prevention of its further progression. Development of such a reliable and efficient software system is possible using a suitable system development life cycle (SDLC) model only. A SDLC model develops a system in a structured, deliberate and methodical mode and provides a very reliable and efficient system within limited resources and time. Macular edema is the blurring or loss of central vision which is caused as a result of Diabetic Retinopathy and Analysis of OCT images helps in identification of Macular Edema. The aim of this research is the successful detection of Macular Edema using Iterative Prototyping SDLC model. First the extraction of ILM layer has been done by using Active Contour based Segmentation and Curve Fitting Techniques then a new technique is proposed in this research for the successful localization of fovea in retinal ILM layer by using distance based method. Finally the detection of Macular edema has been done on the basis of analysis of fovea region. The system is evaluated using a local dataset of OCT images which is gathered with the help of Armed Forces institute of Ophthalmology. The dataset consists of 550 images and the developed system gives an accuracy of 84%.

Keywords—SDLC, Iterative Prototyping, Fovea Centrals, Optical Coherence Tomography, Macula, Macular Edema.

I. INTRODUCTION

A System Development Life Cycle (SDLC) is a methodology which is used to represent the process of development of a system and the system is developed in a structured, deliberate and methodical mode which does reiterate every single stage of the Life Cycle. It is the development of a system or an application following the process of planning, creating, testing and deploying for the success of system under development. There are multiple SDLC models like waterfall model, iterative model and incremental model etc. and to ensure the success of system developed, the perfect model to follow is decided. Figure 1 shows different phases of a Life Cycle Model.

The most suitable model is selected on the basis of Type of system to be developed, Requirements and Functionalities of the system, Skills and Experience. A suitable SDLC is a key to the success of project.

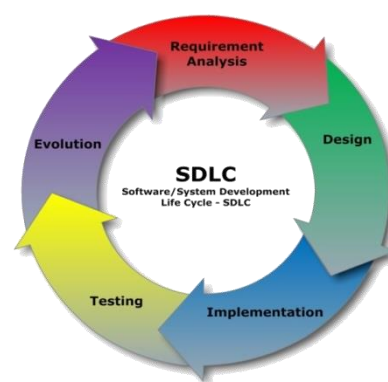


Figure 1: System Development Life Cycle Phases

Rapid prototyping is the type of SDLC model in which quick prototypes are delivered for assessment and after doing modifications, a new prototype is again developed according to new requirements. Edema detection system is aimed to provide highly accurate Edema detection and for that it is necessary to get quick and frequent response from a specialist, during development, who will be using this system. Rapid Prototyping Model is the one which can provide help in development of a system in this way. Hence the development of this system is done using Iterative Prototyping SDLC model.

Macular Edema is the disease in which the macular part of eye gets damaged causing central vision blurring and in severe case the loss of vision. Macular edema is caused due to the leakage of fluids out of blood vessels in retina. The fluids leakage out of blood vessels causes swelling and thickening of Macula and as a result loss of detailed vision. Figure 2 (top) shows the macular edema pointed out in an OCT image in form of cystic pockets and rise in Fovea and Figure 2 (bottom) shows a fundus image with edema in yellow color. Since Macular Edema causes prominent changes in fovea region in OCT that's why Macular Edema can be identified through the localization of fovea region and after that analysis of fovea region in an OCT image. On the basis of changes occurring in fovea region, Macular Edema can be easily identified.

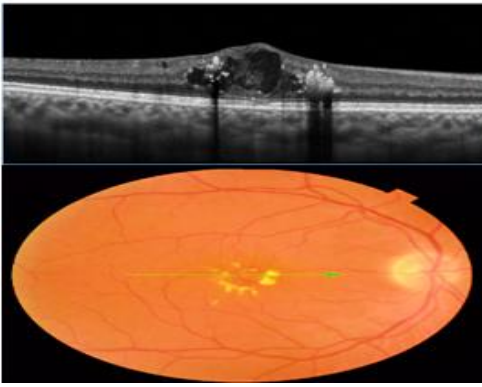


Figure 2: Macular Edema Identification in OCT and Fundus Image

The paper proposes a novel method for identification of Macular Edema in Retinal OCT images using Iterative prototyping SDLC model. Section-I is an introduction of Edema and SDLC model followed by section-II having related work done for Fovea and Edema detection. Section-III is the detailed Methodology section having step by step development of edema detection system. Section-IV describes Results obtained by the system developed and finally Section-V contains conclusion and an overview of the complete system development process.

II. RELATED WORK

There is no work done already for Fovea and Edema detection using OCT images but there are many algorithms proposed by many researchers for fovea and edema detection in Digital Fundus Images [13-16]. One is the detection of fovea in fundus images is by using some morphological operation of Image processing. It finds out the location of optic disk first and then identifies the location of fovea. Since fovea is the non-vascular region of fundus image that's why it is the dark most part of the image. In this way a collection of pixels located at the center and dark colored are marked as fovea [1].

Spatial domain filtration is another method which is used for the detection of fovea. Various gray scale image operations are applied on the fundus image first. Then spatial filtration is applied to extract the macular region. After that fovea is identified as the lowest frequency values in the Macula [2].

Since fovea is the non-vascular region of macula and does not have any blood vessels. By using this property of fovea another technique has been proposed for fovea identification. It identifies the presence or absence of fovea on the basis of blood vessels presence or absence. The thickness of blood vessels is calculated in macula and the region with minimum thickness is considered as fovea [3].

Another method for detection of fovea in fundus images is with the help of color bands. A moving window calculates the average color intensity of the image after the extraction of red and green components of fundus image. The window having minimum average intensity value is marked as fovea. These intensities are calculated after the fusion of red and green components [4].

By using the properties of blood vessels and information of optic disk in fundus image, another algorithm based upon optic disk is also proposed [5]. Graphically represented intensities of color bands are also helpful in the detection of optic disk and fovea region [6].

The extraction of retinal layers is also very important before fovea detection in OCT images and there are many methodologies proposed for this. One method is the assigning of normal feature values for all the layers. Then for a given image, scanning is done vertically. When the values of features start to deviate from present values then it is checked in the next layer. If this condition meets then it is marked as the separation mark between the two layers [7].

Probabilistic model is another one for the extraction of the layers. A probabilistic value for each pixel in the layer is calculated after some preprocessing. The value of layer defines the probability of the pixel to belong to that layer. If the value of layer is high then the pixel will more probably belong to that layer. This is done by using random forest classifier which separates it into layers [8]. Table 3.1 shows an overview of all the related work done.

Due to tilted OCT images during scanning, it becomes sometimes difficult to process the image easily so the alignment of image is very necessary. The flattening of layers aligns them to the x-axis and then bilinear interpolation can be easily used for image alignment [9] [10]. One method for the detection of Macular Edema in fundus images is through the detection of exudates in fundus images. In this method first of all the detection of Optic Disk is very necessary and it has been done by the use of morphological filtration techniques and watershed transforms. After that exudates have been found by their high variations in their gray levels and morphological re-construction techniques have been used to determine their contours. Finally Macular Edema detection has been done on the basis of exudates presence [11].

In another method for the detection purpose of Macular edema it is necessary to find the presence of all of the possible exudates on retinal surface. In this method the exudates have been enhanced with the help of Gabor filtration bank and a binary mask has been generated for the possible locations of exudates. Hence the Macular edema has been identified on the basis of number and locations of exudates on the surface of retina in colored fundus images [12].

III. PROPOSED METHODOLOGY

The whole project development is divided into multiple modules and their sub modules. A prototype module is developed for each module and is handed over to a specialist for feedback. The changes mentioned by the specialist are noted and another prototype is developed on the basis of changes identified. Once the prototype of the changed requirements is completed, it is again handed over to the specialist for feedback and this process goes on as long as the physician gets satisfied with the prototype delivered. This process is repeated for all individual modules. After the entire prototype modules are approved, the final development is

done for the improvement of quality measures of the system. In this way Rapid Prototype is helping in the development of a highly reliable system which is being verified by the user after each single step and the error cost is reducing very much since all the errors are being removed with the development of every next prototype.

Following steps are to be followed during the development of system:

- Requirements Elicitation
- Requirements Specifications
- Architectural Design
- Implementation
- Testing
- Feedback form Specialist
- Improved Prototype Systems development until approval from user
- Improvement of Quality of Final prototype
- Deployment
- Maintenance

Figure3 shows an overview of flow of development from requirements gathering to the completion and maintenance of system.

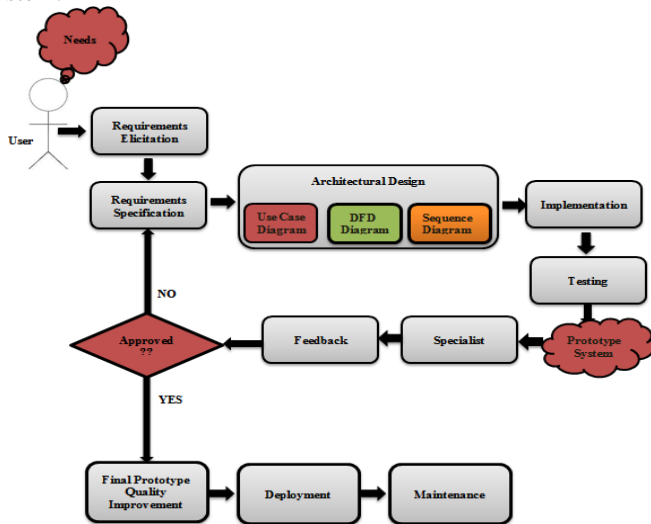


Figure 3: Flow Diagram of the Algorithm

A. Requirements Elicitation

Requirements Elicitation is the first most step of any project development. In this step maximum number of possible requirements are tried to collect. These requirements are not in a well written and structured form but this is the step where most of the information is collected about the system under development. The requirements for this project are collected from the Ophthalmological Department of AFIO. Out of multiple techniques of requirements elicitation e.g. Brainstorming, Interviewing, Questionnaires etc., the requirements for this project are collected through the Interviewing technique. Many interviews are conducted with the potential users and stakeholders of the project and

maximum numbers of requirements are collected for the project. Information shown in Figure4 about the product has been collected after all the interview sessions.

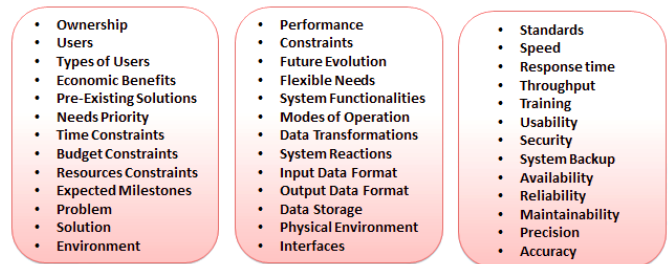


Figure 4: Requirements Elicitation Information Gathered

B. Requirements Specification

In requirements Specification stage requirements roughly gathered are studied deeply and their feasibility is estimated and on the basis of feasibility of each requirement three different categories are defined for a structured view of requirements shown in Figure5. Three types of requirements are:

- Functional Requirements
- Non-Functional Requirements
- Constraint Requirements

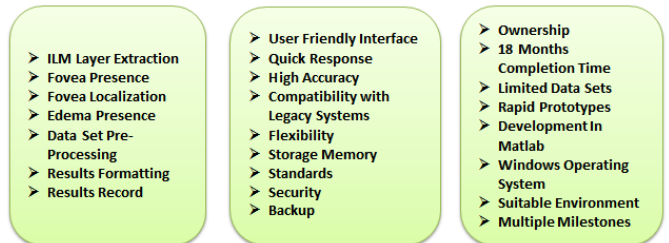


Figure 5: (Left) Functional Requirements; (Middle) Non-Functional Requirements; (Right) Constraint Requirements

C. Architectural Design

Architectural design consists of high level and low level design of the system. It represents the whole system as a model. The model can be in different forms like Use Case diagram, Data Flow Diagram, Sequence Diagram etc. These diagrams are used for the understanding of the system. These diagrams also help in clarifying different confusions about any functionality or any other thing. These diagrams are given to the users of the system for an overview of the system to be designed and if there is any misunderstanding then it is modified at this point. Hence saving the time, effort and resources. Following are some design diagrams used for the understanding and overview of the system:

- Use Case Diagram
- Data Flow Diagram
- Sequence Diagram

Use case diagram is helpful in understanding of the requirements as it is a form of user and system interaction and each use case depicts a scenario of interaction between the user of the system and the system. The user/actor can be a person and can also be some other system. Figure6 is a use

case diagram showing the whole system and its interaction with users.

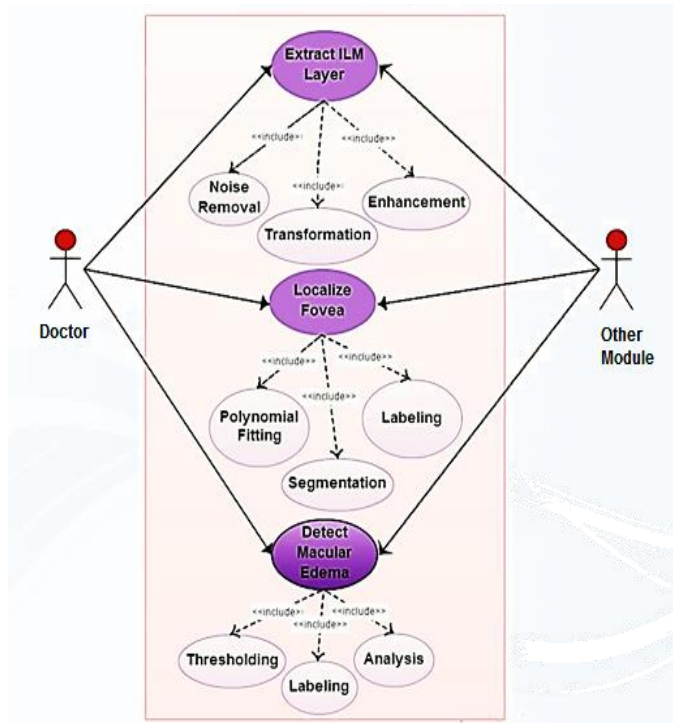


Figure 6: Edema Detection Use Case Diagram

Data flow diagrams show the flow of data between different units and processes. It helps in understanding the project in terms of data processing. It also helps in testing phase as it keeps track of the data flowing between nodes hence an error cause can be estimated. Figure7 shows the data flow diagram of the system.

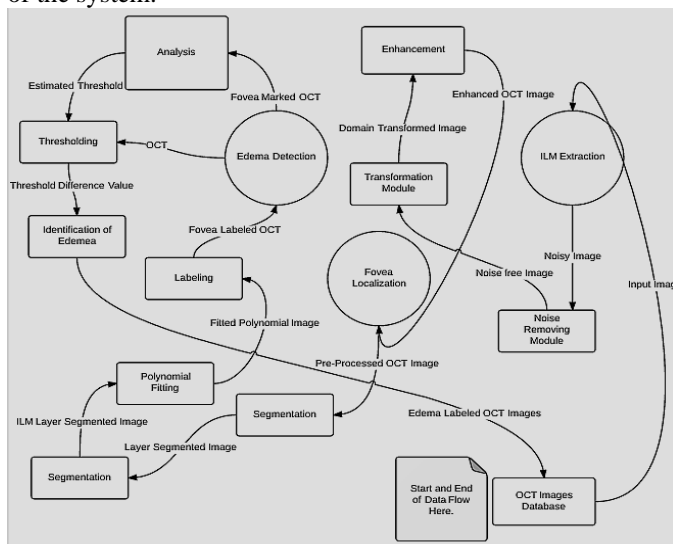


Figure 7: System Data Flow Diagram

A sequence diagram shows the occurrences of different processes at specific intervals of time from start to end of a process. It also shows the life time of different sub processes. Figure8 shows the sequence diagram of the system developed.

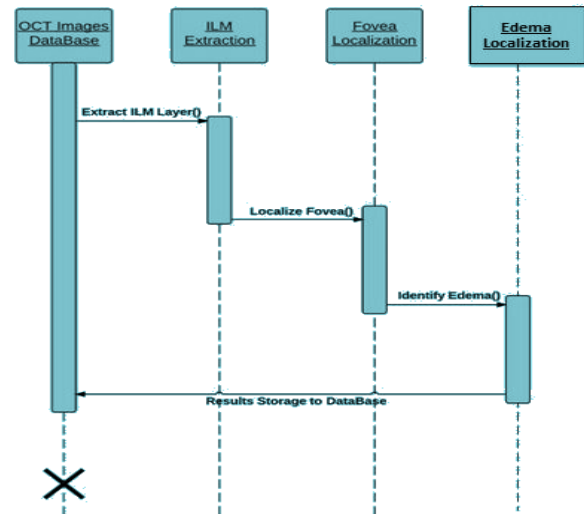


Figure 8: System Sequence Diagram

D. Implementation, Feedback, Modifications

The whole project is divided into three modules.

1. ILM extraction
2. Fovea Localization
3. Edema Detection

There are also sub modules of each main module. Figure9 shows the division of whole project into different modules. Since the modules are not independent but are each next module is depending upon the previous one. That's why three sub modules of main module are developed one by one. First of ILM Extraction module is developed by rapid prototyping. After 4-5 prototypes this module has been approved and finalized. After that the second module is developed by rapid prototyping. All the errors and bugs are removed with each next prototype. Similarly final module has been approved and finalized after some prototypes. Fovea localization and edema detection algorithms are changed many times as the results were either not good or were not acceptable by user. Rapid prototyping has helped in finally achieving a perfect system which is acceptable and according to the needs of user.

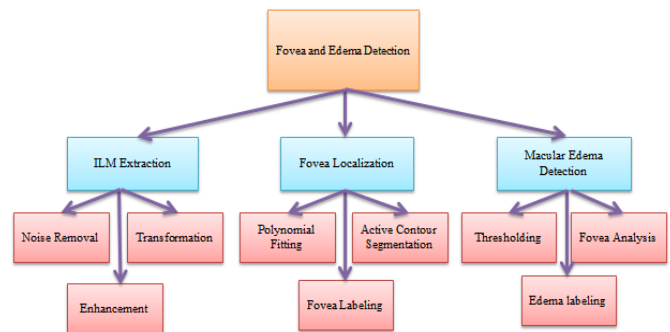


Figure 9: Modules Divisions

The main aim of the system under development is the conversion of an OCT image into a Fovea marked image and after that the labeling of that image as Edema or Non-Edema scan. Following processing is done from Input image to the

conversion of Output image. Figure10 is an overview of Methodology steps.

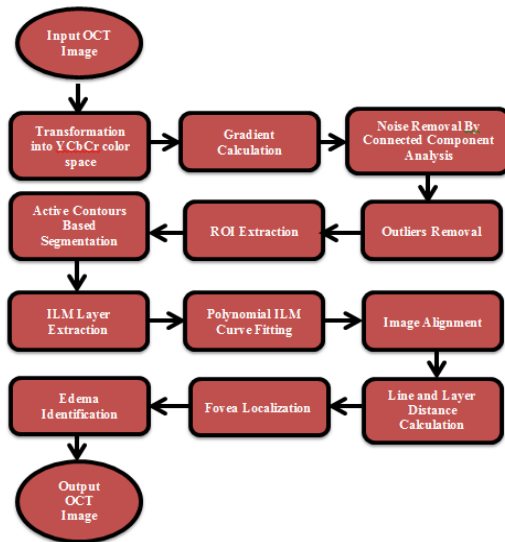


Figure 10: Implementation Overview

Since Macular edema causes prominent changes in top OCT layer of retina which contains fovea so for the purpose of Edema detection, Fovea is to be localized first and for that Extraction of region of Interest (ROI) is the main objective first. Hence an Input OCT image is taken as shown in Figure11(a). Then the input image is converted into YCbCr color space in Figure11(b) to get prominent layers boundaries. In the next step gradient of each color component is calculated separately using Prewitt operator as each color component contributes in the layer boundary formation. After that gradients of all three color components are combined and Thresholding is applied to remove background and much Noisy small objects are removed by performing Connected Components Analysis. Further Noise is removed by taking distance between centroids of each object and extra noisy small objects are removed on the basis of minimum threshold distance between objects. Small objects at a distance greater than threshold are removed. Figure11(c) shows the centroids of each object marked in blue and Figure11(d) shows noise free ROI extracted from original input grayscale image. This ROI is further used as an initial Mask for Active Contours Based Segmentation. This segmentation provides fine layer boundaries out of input grayscale image. Figure11(e) shows the segmented layers region. After that the upper and lower layer boundaries are separated out of segmented image and smoothing of boundaries is done by performing Polynomial Curve Fitting. Figure11(f) shows the upper and lower layer boundaries. Next the alignment of layers is done by drawing a straight line between two extreme points of layers. It helps in alignment of tilted OCT images. Since Edema is the swelling in macula which results in a Rise in macular region of OCT. Point by Point distance between upper layer and Straight line drawn is taken and the maximum distance between these two is noted. If the maximum distance value is greater than a specific threshold and lies in the middle area of layer than it is

marked as an Edema otherwise as Non-edema. Figure11(g) shows the aligned and Edema Marked OCT Image.

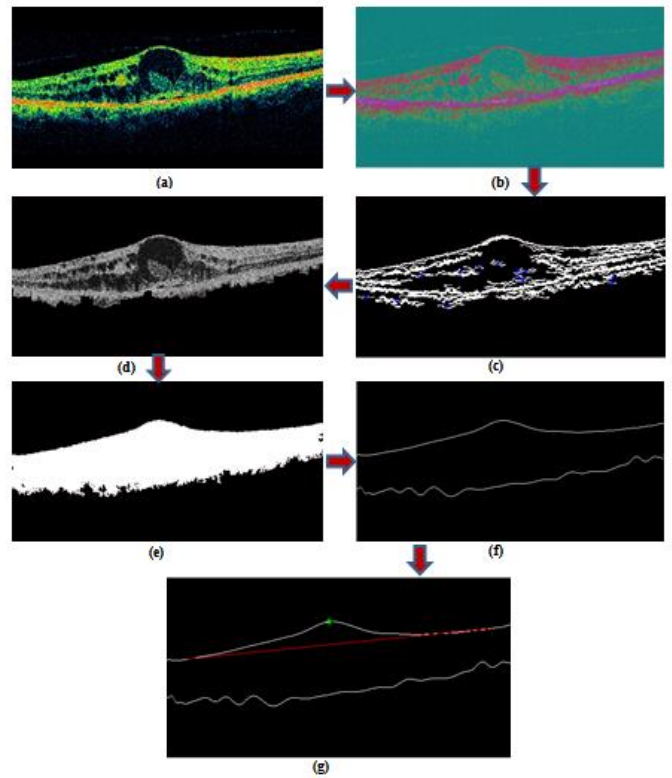


Figure 11: (a) Input Image, (b) Transformed Image, (c) Centroids Labeled, (d) ROI, (e) Segmentation, (f) Layers Boundaries, (g) Edema Labeled Output Image

E. Quality Improvement, Deployment, Maintainance

White Box testing of the system developed is performed by Statement Coverage and Boundary Value Analysis is used for Black Box testing i.e. the testing of each module and each sub module. In boundary value analysis, each functionality is tested by three input values. One is the normal Image, Second is abnormal and third one is poor quality and corrupt image. In this way all the functionalities are first unit tested for these three kind of images. After that sun modules are combined and integration testing is done. Finally System testing is done for the whole system. This testing phase is repeated after every single prototype development for the high quality and bug free system achievement. Since in many cases the Rapid prototyping model does miss some phases like testing mostly to provide the prototype quickly and making the quality of product risky but in this project testing is done after the development of each single prototype to keep the system totally error free. Also the performance testing of final product after prototyping phase completion is done by its usability testing, stress testing, Recovery Testing etc. to increase the quality of end product.

Next thing is the quality improvement of the final product which is improved in terms of response. After this the system has been deployed and the integration with legacy system is improved in the maintenance phase.

IV. RESULTS

Since there are no Datasets available for such system, that's why the OCT images used for the testing of Edema Identification are collected from AFIO. First the images are manually marked by some Specialist and then these images are labeled by using this system. Finally the results of both images are compared and hence the performance and accuracy of system is calculated. A local dataset of 50 images is collected for testing. The dataset contains both with Edema and without Edema presence images in which 15 images are normal and 35 are with Edema. This system correctly classifies 14 images normal and 28 images of Edema and hence the accuracy of system is 84% with 0.93 Sensitivity value and 0.8 Specificity value. Figure12 shows the results of some randomly selected images from dataset.

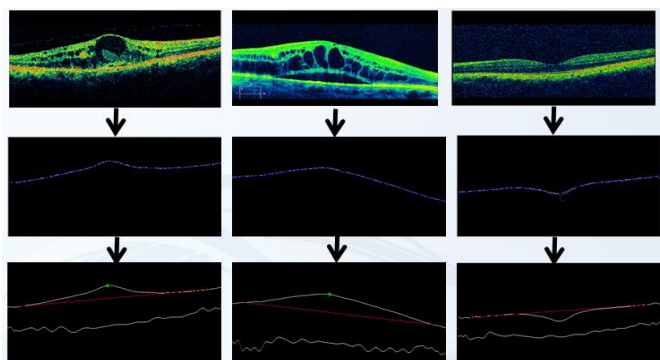


Figure 12: (Top Row) Input Images, (Middle Row) Smooth Top Layer Extracted, (Bottom Row) Edema Localized OCT Images

V. CONCLUSION & DISCUSSION

SDLC is the step by step development of a system to get an Efficient and Reliable system. It helps in Organizing and Efficiently managing the development life cycle to achieve the desired product within available Time and Resources. OCT imaging is a new technique for the detailed imaging of retina. Analysis of OCT images is helpful in detection of multiple diseases as the changes occurring in the retinal layers due to some disease can be easily observed with OCT image analysis. Macular Edema is the disease in which the sharp and pin-point vision gets affected. Since there is no work already done for the detection of such things using OCT imaging so in this research a new technique is proposed for the detection of Edema in OCT images using an appropriate SDLC model. It works on the principle of line-layer distance calculation and the labeling of Edema at the maximum positive distance point. Before that the upper most retinal layer is successfully extracted out of OCT image which is the key element for further Edema Identification processing. The extraction of ILM layer is done by using Active contour Based segmentation and polynomial Curve fitting techniques. Similarly the identification of Edema is done on the basis of rise in fovea region from a specific threshold. This system is further tested for a data set of 50 images and Edema detection gives an accuracy of 84%.

ACKNOWLEDGMENT

We might want to recognize the assistance and exceptionally kind support of AFIO (Armed Forces Institute of Ophthalmology), Rawalpindi. We additionally say thanks to National ICT R&D fund, Pakistan for their monetary support.

REFERENCES

- [1] Asim, K.M., Basit, A. , Jalil, A.. "Detection and localization of fovea in human retinal fundus images," 2012 International Conference on Emerging Technologies (ICET), 2012.
- [2] Guven, A. ; Oner, A.O. ; Kara, S. "Automated location of optic disk and fovea in color fundus images" 14th National Biomedical Engineering Meeting, 2009. BIYOMUT 2009.
- [3] Ziyang Liang ; Wong, D.W.K. ; Jiang Liu ; Ngan-Meng Tan ; Xiangang Cheng ; Cheung, G.C.M. ; Bhargava, M. ; Tien Yin Wong "Automatic fovea detection in retinal fundus images" Industrial Electronics and Applications (ICIEA), 2012.
- [4] Veras, R., Silva, R. ; Aires, K. ; Medeiros, F. "Automatic Detection of Fovea in Retinal Images Using Fusion of Color Bands" Graphics, Patterns and Images (SIBGRAPI), 2014.
- [5] Kovacs, L. ; Qureshi, R.J. ; Nagy, B. ; Harangi, B. ; Hajdu, A. "Graph based detection of optic disc and fovea in retinal images" Soft Computing Applications (SOFA), 2010.
- [6] Samanta, S; Saha, S.K. ; Chanda, B. "A Simple and Fast Algorithm to Detect the Fovea Region in Fundus Retinal Image" Emerging Applications of Information Technology (EAIT), 2011.
- [7] Yogesh Kumar A., Sasikala M "Texture Analysis of Retinal Layers in Spectral Domain OCT Images" International Journal of Emerging Technology and Advanced Engineering Volume 2, Issue 12, December 2012.
- [8] Andrew Lang, Aaron Carass, Elias Sotirchos, Peter Calabresi, and Jerry L. Prince "Segmentation of retinal OCT images using a random forest classifier" Proc SPIE. March 13, 2013.
- [9] Yang Q, Reisman CA, Wang Z, Fukuma Y, Hangai M, Yoshimura N, Tomidokoro A, Araie M, Raza AS, Hood DC, Chan K. "Automated layer segmentation of macular OCT images using dual scale gradient information". Opt Express. 2010.
- [10] Garvin M, Abramoff M, Wu X, Russell S, Burns T, Sonka M. "Automated 3-D intra retinal layer segmentation of macular spectral-domain optical coherence tomography images". IEEE Trans MedImag. 2009.
- [11] Thomas Walter, Jean-Claude Klein, Pascale Massin, and Ali Erginay "A Contribution of Image Processing to the Diagnosis of Diabetic Retinopathy—Detection of Exudates in Color Fundus Images of the Human Retina" IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 21, NO. 10, OCTOBER 2002.
- [12] Umer Aftab and M. Usman Akram "Automated Identification of Exudates for Detection of Macular Edema" 2012 Cairo International Biomedical Engineering Conference (CIBEC) Cairo, Egypt, December 20-21, 2012.
- [13] M. U. Akram and S. A. Khan, "Automated detection of dark and bright lesions in retinal images for early detection of diabetic retinopathy", Journal of Medical Systems (JOMS), vol. 36, no. 5, 3151-3162, 2012.
- [14] A. Tariq, M. U. Akram, A. Shaukat, S. A. Khan, "Automated Detection and Grading of Diabetic Maculopathy in Digital Retinal Images", Journal of Digital Imaging, vol. 26, no. 4, pp. 803-812, 2013.
- [15] M. U. Akram, A. Tariq, M. A. Anjum, M. Y. Javed, "Automated Detection of Exudates in Colored Retinal Images for Diagnosis of Diabetic Retinopathy", OSA Journal of Applied Optics, vol. 51 no. 20, 4858-4866, 2012.
- [16] M. U. Akram, S. Khalid, S. A. Khan, "Identification and Classification of Microaneurysms for Early Detection of Diabetic Retinopathy", Pattern Recognition, vol. 46, no.1, 107-116, 2013.

A metrics-based comparative study on object-oriented programming languages

Di Wu

Lin Chen

Yuming Zhou

Baowen Xu *

State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, China

nju.wudi@gmail.com

lchen@nju.edu.cn

zhouyuming@nju.edu.cn

bwxu@nju.edu.cn

Abstract—There has been a long debate on which programming language can help write better object-oriented programs. However, to date little response is given to this issue with empirical evidence. In this paper, we perform a comparative study on C++, C#, and Java programs by using object-oriented metrics, which comprise measures for class size, complexity, coupling, cohesion, inheritance, encapsulation, polymorphism, and reusability. Our experiment is conducted on 78 tasks in Rosetta Code, a code repository providing solutions to the same programming tasks in different languages. The experimental results show that: (1) C++ classes are significantly larger than C# and Java classes in size, but their complexity does not differ significantly; (2) C# classes are significantly more likely to be coupled than C++ and Java classes through inter-class method invocations instead of direct data access; (3) C# and Java classes tend to be more cohesive than C++ classes; (4) C# and Java significantly outperform C++ in building deep inheritance trees; and (5) programs written in C++, C#, and Java do not show a significant difference in class encapsulation, polymorphism, and reusability. These findings could help practitioners choose suitable languages to develop object-oriented systems.

Keywords- Programming Language, Comparative Study, Object-oriented Metrics

I. INTRODUCTION

Which programming language can help write better object-oriented programs? This question is often asked but it is hard to reach a consensus on the answer. From the practical perspective, it would be reliable to answer this question by empirically comparing real object-oriented programs written in different languages. The findings based on empirical evidence should be valuable in helping practitioners choose suitable languages to develop object-oriented systems.

To evaluate the quality of object-oriented programs, many metrics have been proposed, which are related to various language features like class size, coupling, cohesion, inheritance, encapsulation, and polymorphism [4-9]. In previous studies, these metrics are generally applied to fault prediction [10], class testability prediction [11], code refactoring [12], and code size estimation [13]. However, few researchers use the object-oriented metrics as indicators to compare programs written in different languages.

In this paper, we perform a preliminary comparative study on programming languages by employing 23 commonly-used

object-oriented metrics. More specifically, we use the standard statistical inference techniques to perform a differential analysis on the metric values for real programs written in C++, C#, and Java. The subject programs used in this study are provided by Rosetta Code [21], a code repository of solutions to common programming tasks implemented with various languages. By investigating 78 tasks in Rosetta Code, we attempt to answer the following issues: (1) Which language can help write classes of small size and low complexity? (2) Which language can help write classes of low coupling and high cohesion? (3) Which language can help create good type hierarchies? and (4) Which language can help write classes of good encapsulation, polymorphism, and reusability? These issues are of highly practical value, as they determine which programming language can help write better object-oriented programs. However, little is currently known on this subject with empirical evidence. Our study attempts to fill this gap by this comparative study.

Our experimental results based on object-oriented metrics show the following findings:

- C++ classes are significantly larger than C# and Java classes in size, but their complexity does not differ significantly;
- C# classes are significantly more likely to be coupled than C++ and Java classes through inter-class method invocations instead of direct data access;
- C# and Java classes tend to be more cohesive than C++ classes;
- C# and Java significantly outperform C++ in building deep inheritance trees;
- Programs written in C++, C#, and Java do not show a significant difference in class encapsulation, polymorphism, and reusability.

The rest of the paper is organized as follows. Section II introduces the object-oriented metrics in a nutshell. Section III describes the studied subjects, data collection procedure, and data analysis method. Section IV reports the experimental results. Section V presents the threats to validity. Section VI discusses related work. Section VII concludes the paper and outlines the direction for future work.

II. OBJECT-ORIENTED METRICS

In the past decades, many object-oriented metrics have been proposed. The most well-known metrics are CK metrics [4] and MOOD metrics [6], which are applied to assess the quality

* Corresponding author: Baowen Xu; Email: bwxu@nju.edu.cn

TABLE I. OBJECT-ORIENTED METRICS

Category	Metric name	Metric definition	Level	Expected value	Source
Size and Complexity	NOM (Number of methods)	The number of methods defined in a class	Class	Low	[7]
	NOA (Number of attributes)	The number of attributes defined in a class	Class	Low	[7]
	WMC (Weighted method complexity)	The sum of complexity for all methods in a class	Class	Low	[4]
	CC (Class complexity)	The sum of complexity for all methods in a class based on the information flow	Class	Low	[9]
Coupling	RFC (Response for a class)	The number of methods that can be potentially executed in response to a message received by an object of a class	Class	Low	[4]
	CBO (Coupling between objects)	The number of other classes to which a class object is coupled	Class	Low	[4]
	DAC (Data abstract coupling)	The number of ADT(Abstract Data Type) instances defined in a class	Class	Low	[7]
	MPC (Message passing coupling)	The number of send statements defined in a class	Class	Low	[7]
	CF (Coupling factor)	$CF = \frac{\sum_{i=1}^{TC} \sum_{j=1}^{TC} is_client(C_i, C_j)}{TC^2 - TC}$, where TC is total number of classes and $is_client(C_i, C_j) = \begin{cases} 1, & \text{if } C_i \Rightarrow C_j \text{ and } C_i \neq C_j \\ 0, & \text{otherwise} \end{cases}$	System	Low	[6]
Cohesion	LCOM (Lack of cohesion in methods)	LCOM = (Number of pair of methods that have no common attributes) - (Number of pair of methods that have common attributes)	Class	Low	[4]
	TCC (Tight class cohesion)	TCC = (Number of pairs of directly connected public methods using common attributes) / (Number of pairs of public methods)	Class	High	[5]
	LCC (Loose class cohesion)	LCC = (Number of pairs of directly and indirectly connected public methods using common attributes) / (Number of pairs of public methods)	Class	High	[5]
	ICH (Information based cohesion)	The number of invocations to other member functions/methods	Class	High	[8]
Inheritance	NOC (Number of children)	The number of immediate subclasses of a class in a type hierarchy	Class	High	[4]
	DIT (Depth of inheritance tree)	The maximum length from the node to the root of the tree	Class	High	[4]
	MIF (Method inheritance factor)	MIF = (Number of methods inherited in all classes) / (Number of methods defined and inherited in all classes)	System	High	[6]
	AIF (Attribute inheritance factor)	AIF = (Number of attributes inherited in all classes) / (Number of attributes defined and inherited in all classes)	System	High	[6]
Encapsulation	MHF (Method hiding factor)	Let V (M) = number of classes where the method M is visible, then $MHF = 1 - \frac{\sum V(M)}{\text{total number of classes} - 1}$	System	High	[6]
	AHF (Attribute hiding factor)	Let V (A) = number of classes where the attribute A is visible, then $AHF = 1 - \frac{\sum V(A)}{\text{total number of classes} - 1}$	System	High	[6]
Polymorphism	NMO (Number of methods overridden by a subclass)	The number of methods in a subclass overridden from its base class	Class	High	[7]
	PF (Polymorphism factor)	$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}$, where TC is the total number of classes and $M_n(C_i)$ = Number of new methods of the class C_i , $M_o(C_i)$ = Number of overriding methods of the class C_i , $DC(C_i)$ = Number of descendants of the class C_i	System	High	[6]
Reusability	RR (Reuse ratio)	RR = (Total number of super classes) / (Total number of classes)	System	High	[7]
	SR (Specialization ratio)	SR = (Total number of sub-classes) / (Total number of super classes)	System	High	[7]

of object-oriented programs at different levels. To be specific, CK metrics are mainly used to evaluate single classes, while MOOD metrics are applied to assess the whole object-oriented systems. Table I gives a detailed description of the 23 commonly-used object-oriented metrics. According to this table, all the metrics can be divided into 7 categories [3], which cover the following object-oriented features:

- Size and complexity. NOM and NOA are used to measure the size of a class in terms of the number of methods and the number of attributes, respectively. WMC and CC are applied to measure the complexity of a class through

calculating the total complexity of its member functions/methods in different ways. Since classes are suggested to be designed as concise as possible, these metrics are expected to be low in their values.

- Coupling. Five metrics are used to evaluate class coupling from different perspectives. To be specific, the CF metric is used to evaluate the coupling of all classes at the system level. By comparison, the other four metrics measure coupling at class level. Among these metrics, RFC and MPC are used to assess method coupling, DAC embodies data coupling between classes, and CBO shows coupling

between class instances. Since highly coupled classes are less object-oriented, low metric values are preferable.

- Cohesion. Cohesion is measured with four class-level metrics, which are calculated in different ways to reflect the interactions between member functions/methods. Among these metrics, a low LCOM value is expected, while high TCC, LCC, and ICH values are desired.
- Inheritance. NOC and DIT are class-level metrics, which express class inheritance through the number of descendants and the depth of type inheritance, respectively. By comparison, MIF and AIF are system-level metrics, which refer to method inheritance and attribute inheritance, respectively. Since it is suggested to build hierarchical type trees in the object-oriented systems, the high inheritance metric values are expected.
- Encapsulation. MHF and AHF are indicators to show how well methods and attributes are hidden inside classes. These metrics are measured at system level and high metric values are preferable.
- Polymorphism. NMO and PF are polymorphism metrics at different levels. To be specific, NMO is a class-level metric, which refers to the number of methods overridden by a single subclass, while PF is a system-level metric, which measures the degree of method overriding in the whole type tree. Their metric values are desired to be high.
- Reusability. RR and SR are both system-level reusability metrics. They are calculated as the ratios of subclasses to all classes and to super classes, respectively. Since classes are expected to be highly reused, large reusability metric values are desirable.

III. RESEARCH METHOD

In this section, we first introduce the subject programs used in our study. Then, we describe the data collection procedure. Finally, we show the data analysis method.

A. Studied Subjects

In order to conduct the comparative experiment, we need to investigate the programs that give solutions to the common goals and are written in C++, C#, and Java, respectively. For this reason, we employ the open-source programs in Rosetta Code [21], a code repository providing solutions to the same tasks in various languages. Currently, it contains 766 programming tasks implemented in 567 different languages. These tasks belong to 59 categories, including mathematics, games, and networking, etc. Due to its abundant resources, Rosetta Code has been effectively used to compare languages' concise, performance, and failure-proneness [2].

In the Rosetta Code repository, we totally find 381 tasks that have solutions in all the three investigated languages. By manually checking these solutions, we choose 78 tasks as our studied subjects, because they are all implemented in the object-oriented manner. In other words, the remaining 203 tasks are deleted from our concern either because they are implemented in the procedural manner in their C++ solutions or because that they are lack of entire implementation code. The detailed information of these tasks can be found at <http://ise.nju.edu.cn/wudi/Lang.Comp.Study>.

B. Data Collection

We collected the metric values by using "Understand" [20], a program analysis and measurement tool. Specifically, the data was collected by the following steps. At the first step, we built an Understand database for each solution implemented in C++, C#, and Java. At the second step, we collected the metric values for each solution by processing its database. Some simple metrics such as NOM, NOA, and WMC were directly reported by Understand, while other metrics including CC, MHF, and AHF were collected by running our own Perl scripts, which utilize the analysis-based information of programs through calling Understand APIs. At the third step, we calculated for each class-level metric its average metric value of all classes in each solution. At the fourth step, we selected for each task its optimal solution written in the same language. Of the 78 studied subjects, 20 tasks have more than one solution written in the same language. In order to pick out the best solutions for the 20 tasks, we compare for each task its solutions written in the same language according to the metric values and select the optimal one. At the last step, we gathered the metric values of all selected solutions to the 78 tasks and stored the data in a csv file, which was used for data analysis.

C. Data Analysis

We employ the standard statistical inference techniques to analyze the experimental data. More specifically, for each object-oriented feature, we perform a Wilcoxon's signed rank analysis to compare the metric values of solutions implemented in different languages. In other words, C++, C#, and Java are compared in pair-wise to find out which language can help write best object-oriented programs. If the metric values of two languages show a difference at a significance level of 0.05 (p-value), we will conclude that the languages are significantly different. Also, we employ the Cliff's δ to examine whether the magnitude of difference is important [18]. By convention, the magnitude of the difference is considered either trivial ($|\delta| < 0.147$), small (0.147-0.33), medium (0.33-0.474), or large (> 0.474) [19]. Finally, we apply the signed ratio R to give an unstandardized measure of the difference between two medians [2]. The R value is calculated as:

$$R = \text{sgn}(M_x - M_y) \frac{\max(M_x, M_y)}{\min(M_x, M_y)} \quad (1)$$

where M_x and M_y denotes the median metric values of language X and language Y, respectively. A positive sign $\text{sgn}(M_x - M_y)$ indicates that the median metric value of X is larger than the median metric value of Y, while a negative sign signifies a reverse result. Moreover, the absolute R value denotes how many times X's median is larger/smaller than Y's median under a specific metric.

VI. EXPERIMENTAL RESULTS

In this section, we report in detail the experimental results. Table II shows the overall experimental results for language comparison. In this table, we present for each metric the significance of the Wilcoxon's signed rank analysis (p-value), the magnitude of difference (Cliff's δ), and the times between two median values (R).

TABLE II. COMPARATIVE RESULTS FOR OBJECT-ORIENTED METRICS ON C++, C#, AND JAVA PROGRAMS

Metrics		C++ vs. C#			C++ vs. Java			C# vs. Java		
		<i>p</i>	δ	<i>R</i>	<i>p</i>	δ	<i>R</i>	<i>p</i>	δ	<i>R</i>
Size and Complexity	NOM	< 0.001	0.478	2.222	< 0.001	0.484	1.667	1.000	-0.007	-1.333
	NOA	0.007	0.385	-	0.554	0.162	2.000	0.064	-0.151	-
	WMC	0.150	0.095	1.200	0.930	-0.024	1.000	0.006	-0.125	-1.200
	CC	0.111	-0.143	-2.773	0.460	-0.092	-1.340	1.000	0.049	2.069
Coupling	RFC	< 0.001	-0.635	-1.403	< 0.001	0.502	2.000	< 0.001	0.868	2.806
	CBO	< 0.001	-0.677	-3.000	< 0.001	0.463	-	< 0.001	0.916	-
	DAC	0.004	0.236	-	0.301	-0.027	-	< 0.001	-0.236	-
	MPC	0.954	-0.029	-	1.000	0.030	-	0.966	0.061	-
	CF	0.072	-0.827	-2.222	0.056	-0.753	-2.778	1.000	0.012	-1.250
	LCCOM	< 0.001	0.458	-	< 0.001	0.340	-	0.416	-0.070	-
Cohesion	TCC	0.351	0.111	1.473	0.884	0.007	-1.018	0.966	-0.095	-1.500
	LCC	0.254	0.155	1.500	0.777	0.055	1.000	0.966	-0.098	-1.500
	ICH	0.014	-0.272	-	0.010	-0.268	-	1.000	-0.001	1.136
	NOC	0.078	0.133	1.000	0.004	0.291	1.000	0.065	0.155	1.000
Inheritance	DIT	< 0.001	-0.934	-	< 0.001	-0.939	-	0.210	-0.068	1.000
	MIF	0.150	0.094	-	0.760	0.052	-	0.378	-0.041	-
	AIF	1.000	0.033	-	1.000	0.034	-	1.000	0.001	-
Encapsulation	MHF	0.218	-0.219	-2.826	0.230	0.375	-	0.118	0.625	-
	AHF	0.608	0.184	2.000	0.385	0.306	-	0.497	0.388	-
Polymorphism	NMO	0.317	-0.028	-	0.569	0.026	-	0.178	0.053	-
	PF	1.000	0.333	-	1.000	0.000	1.000	1.000	-0.111	-
Reusability	RR	0.159	0.088	-	0.056	0.077	-	1.000	-0.012	-
	SR	0.432	-0.625	-1.833	1.000	0.250	1.500	0.378	0.500	2.750

* Note: (1) All p-values have been adjusted using the Benjamini-Hochberg method; (2) Cells marked with “-” denote the denominator of formula (1) is zero; (3) Cells in gray background denote the significant results (p-values < 0.05).

A. Size and Complexity

We employ the result from NOM and NOA metrics to compare the size of classes written in C++, C#, and Java. According to Table II, we find that C++’s NOM value is significantly different from C#’s NOM value ($p < 0.001$), and the magnitude of difference is large in terms of Cliff’s δ (0.478). Moreover, the median NOM value of C++ is over 2 times larger than the median NOM value of C# ($R = 2.222$). Besides, the comparison between C++’s NOM value and Java’s NOM value shows a similar result. This indicates that C++ classes significantly have more member functions/methods than both C# and Java classes. As for NOA, we find C++ classes significantly have more attributes than C# classes ($p = 0.007$) and the magnitude of difference is medium ($\delta = 0.385$). However, the comparison between C++ and Java does not show a significant difference.

In terms of class complexity (WMC and CC metrics), we do not observe a significant difference between C++ and C#/Java. This indicates that C++ classes are not significantly more complex than C# and Java classes. However, C#’s WMC value is significantly different from Java’s WMC value ($p < 0.001$), but the effect size is trivial according to Cliff’s δ (-0.125). This signifies that C# methods tend to be less complex than Java methods, but the difference is not obvious.

To summarize, the core observation from the size and complexity metrics is that **C++ classes are significantly larger than C# and Java classes in size, but their complexity does not differ significantly.**

Interpretation. One possible explanation for this result is that C++ has two different paradigms, namely the procedural programming and the object-oriented programming. When

programmers implement C++ classes, they are likely to think in the procedural manner, thus resulting in a large number of member functions to be created inside a class.

B. Coupling

We employ the result from RFC, CBO, DAC, MPC, and CF metrics to compare the coupling of classes written in C++, C#, and Java. As for CF, the system-level metric, we do not find any significant difference among the three languages (all p-values > 0.05). This indicates that C++, C#, and Java do not differ in class coupling from the system-level perspective. As for the class-level metrics, however, these three languages show significant differences. To be specific, for RFC, which evaluates class coupling based on method invocations, C# has a significantly larger RFC value than C++ and Java (both p-values < 0.001). Moreover, the effect sizes are large in terms of Cliff’s δ ($0.635 \leq |\delta| \leq 0.868$). Besides, the *R* values also show a difference between the medians ($1.403 \leq |R| \leq 2$). This evinces that C# classes are more likely to interact with each other through inter-class method invocations. Also, CBO, a metric reflecting coupling between class objects, shows a similar result. However, DAC, a metric evaluating class coupling through data access, indicates a contrary result. More specifically, it shows that C#’s DAC value is significantly smaller than C++ and Java’s DAC values (both p-values < 0.005). Moreover, the effect sizes are small in terms of Cliff’s δ (both $|\delta| = 0.236$). This result signifies that C# classes are less likely to interact with each other through data interaction. As for MPC, another coupling metric based on member functions/methods, does not show a significant result. To summarize, the core observation from the coupling metrics is that **C# classes are significantly more likely to be coupled**

than C++ and Java classes through inter-class method invocations instead of direct data access.

Interpretation. According to Table II, we find RFC and MPC, the metrics for inter-class method coupling, show completely different results. This is due to the different ways in calculating their metric values. To be specific, the get/set accessors in C# classes are regarded as ordinary methods when we compute the RFC values. But they are removed during calculating the MPC values. For this reason, we conjecture that the tight coupling among C# classes is generally caused by frequent inter-class get/set method invocations.

C. Cohesion

We employ the result from LCOM, LCC, TCC, and ICH metrics to compare the cohesion of classes written in C++, C#, and Java. As for LCOM, we find that C++ has a significantly larger LCOM value than C# and Java (both p-values < 0.001). Moreover, the magnitudes of difference are medium in terms of Cliff's δ ($0.340 \leq |\delta| \leq 0.458$). Since a low LCOM value is preferable, this result indicates that C++ classes are less cohesive than C# and Java classes. ICH, another cohesion metric, shows a consistent result. To be specific, C++'s ICH value is significantly smaller than both C#'s ICH value ($p = 0.014$) and Java's ICH value ($p = 0.010$). Furthermore, the effect sizes are small in terms of Cliff's δ ($0.268 \leq |\delta| \leq 0.272$). However, the other two cohesion metrics, namely LCC and TCC do not show significant results. To summarize, the core observation from the cohesion metrics is that among the four metrics, two of them significantly evince that C# and Java outperform C++ in creating higher cohesive classes. From this reasoning, we conclude that **C# and Java classes tend to be more cohesive than C++ classes.**

Interpretation. One possible explanation for this result is that many member functions in C++ classes are still implemented in the procedural manner. As a result, the member functions do not interact well through sharing the common attributes and thus result in a low cohesion for the whole class.

D. Inheritance

We employ the result from NOC, DIT, MIF, and AIF metrics to compare the inheritance of classes written in C++, C#, and Java. As for NOC, we find C++'s NOC value is significantly larger than Java's NOC value ($p = 0.004$). Moreover, the effect size is small in terms of Cliff's δ (0.291). The comparison between C+ and C#, however, does not show a significant result ($p = 0.078$). As for DIT, we observe that C++'s DIT value is significantly smaller than both C# and Java's DIT values (both p-values < 0.001). Furthermore, the effect sizes are relatively large in terms of Cliff's δ ($0.934 \leq |\delta| \leq 0.939$). As for other two metrics, namely MIF and AIF, no significant result is revealed. To summarize, the core observation from the inheritance metrics is that **C# and Java significantly outperform C++ in building deep inheritance trees (DIT).**

Interpretation. Both C# and Java have the root type "Object", which is a super type for all classes. Therefore, the depth of inheritance trees in C# and Java systems is not surprisingly larger than the C++ systems.

E. Encapsulation, Polymorphism, and Reusability

We employ the result from MHF and AHF metrics for encapsulation, NMO and PF metrics for polymorphism, and RR and SR for reusability to compare the classes written in C++, C#, and Java. According Table II, there is no significant result revealed by all these metrics. For this reason, we conclude that **programs written in C++, C#, and Java do not show a significant difference in class encapsulation, polymorphism, and reusability.**

Interpretation. Regarding the language features for class encapsulation, C++, C#, and Java all provide private, protected, and public keywords to control the accessibility to the methods and attributes inside a class. As a result, there is no difference in information hiding of classes written in different languages. Moreover, all these three languages support static binding and dynamic dispatching in similar ways. Therefore, the polymorphism of classes does not differ. Finally, class reusability is generally independent from language support. Instead, it is determined by the ways programmers define new classes. For this reason, there is not a significant difference in the reusability of classes written in C++, C#, and Java.

VII. THREATS TO VALIDITY

The threat to the construct validity is the correctness of the metric values collected from "Understand" databases. Since many historical studies have produced reliable empirical results by using "Understand" [20], the data in our study can also be considered as acceptable. The threat to internal validity is the object-oriented metrics used in this study. We totally use 23 metrics to investigate class size, complexity, coupling, cohesion, inheritance, encapsulation, polymorphism, and reusability. Even though these metrics do not include all object-oriented metrics proposed in historical studies, they are generally regarded as the most representative ones. Since these metrics are also used in other empirical studies [1, 3], they are also applicable in this study. The threat to the external validity is that we only use the programs provided by Rosetta Code to conduct the experiment. Since many solutions to our studied tasks have a small number of classes, our empirical results need to be further examined on more complex object-oriented systems in the future.

VIII. RELATED WORK

The most related study to our work was undertaken by Kumari and Bhasin [1], who used the object-oriented metrics to compare C++ and Java programs. They investigated 15 object-oriented applications and found that Java is more object-oriented than C++ as per intuition. The metrics applied in our study and in [1] are basically the same. However, we use more strict research method and thus obtain more reliable findings. To be specific, our study has the following advantages. First, we studied three languages, namely C++, C#,

and Java, while in [1], only C++ and Java were analyzed. Second, we used 78 open-source programming tasks to do the experiment, while Kumari and Bhasin only employed 15 tasks. Moreover, the detailed information of their data set was not given, making their experiment not replicable. Third, we used the standard statistical inference techniques (such as the Wilcoxon's signed rank analysis) to analyze the experimental data, while Kumari and Bhasin only got the comparison result based on bar graphs for the raw metric values. For this reason, our empirical results are more reliable. In terms of the conclusions, the main difference between the two papers lies in the languages' support to build deep inheritance trees. In [1], the authors showed that the DIT metric value of C++ programs was larger than Java programs. However, we get an opposite result. Due that the result in this study is drawn using the standard statistical analysis, our conclusion is more acceptable.

Another related study was conducted by Nanz and Furia [2], who used the Rosetta Code repository to compare the conciseness, performance, and failure-proneness of programs written in different languages. By using the standard statistical inference methods, the authors had the following findings: (1) functional and scripting languages are more concise than procedural and object-oriented languages; (2) C is hard to beat when it comes to raw speed on large inputs; and (3) compiled strongly-typed languages are less prone to runtime failures than interpreted or weakly-typed languages. Compared with [2], our study focuses on a different research question, namely languages' support to write good object-oriented programs. For this reason, the conclusions of the two studies are not comparable. However, the two studies still share some similarities. First, they both use Rosetta Code as the experimental subject. Second, they both use the standard statistical inference techniques to analyze the experimental data. Third, the conclusions of both studies are drawn on the result of statistical analysis. Other related empirical programming language research include the comparison on languages' difference in running time, memory consumption, and productivity [14], the survey of developers' behaviors in using object-oriented concepts [15], the study on languages' support for code quality [16], and the investigation on languages' adoption [17], etc.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we perform a comparative study on C++, C#, and Java programs to investigate which language can help write better object-oriented programs. By analyzing 23 object-oriented metrics on the solutions to 78 real programming tasks, we find that C# and Java outperform C++ in creating concise and cohesive classes. Also, the empirical result shows that C# and Java can help build deeper inheritance trees than C++. Moreover, we find that C# classes are significantly more likely to be coupled than C++ and Java classes through inter-class method invocations instead of direct data access. Finally, the statistical result reveals that the programs written in C++, C#, and Java do not show a significant difference in class encapsulation, polymorphism and reusability. Our empirical evidence should be valuable in helping practitioners choose

suitable languages to develop object-oriented systems. In the future work, we will investigate more object-oriented languages and replicate the study on more applications.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (61170071, 61432001, 91418202, 61472175, 61472178), the National Natural Science Foundation of Jiangsu Province (BK20130014), and the program B for Outstanding PhD candidate of Nanjing University.

REFERENCES

- [1] U. Kumari, S. Bhasin. Application of object-oriented metrics to C++ and Java: A comparative study. *ACM SIGSOFT Software Engineering Notes*, 36(2), 2011: 1-10.
- [2] S. Nanz, C. A. Furia. A comparative study of programming languages in Rosetta Code. *ICSE*, 2015.
- [3] K. K. Aggarwal, Y. Singh, A. Kaur, R. Malhotra. Empirical study of object-oriented metrics. *Journal of Object Technology*, 5(8), 2006: 149-173.
- [4] S. R. Chidamber, C. F. Kemerer. A metrics suite for object-oriented design. *IEEE Trans. Software Eng.*, 20(6), 1994: 476-493.
- [5] L. C. Briand, J. W. Daly, J. Wüst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 3(1): 1998: 65-117.
- [6] R. Harrison, S. J. Counsell, R. V. Nithi. An evaluation of MOOD set of object oriented software metrics. *IEEE Trans. Software Eng.*, 24(6), 1998: 491-496.
- [7] B. Henderson-Sellers. Object-oriented metrics: measures of complexity. *Prentice Hall*, 1995.
- [8] Y. S. Lee, B. S. Liang, S. F. Wu, F. J. Wang. Measuring the coupling and cohesion of an object-oriented program based on information flow. *QSIC*, 1995.
- [9] Y. S. Lee, B. S. Liang, F. J. Wang. Some complexity metrics for OO programs based on information flow: A study of C++ programs. *Journal of Information Science and Engineering*, 10(1), 1994: 21-50.
- [10] T. Gyimothy, R. Ferenc, I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Software Eng.*, 31(10), 2005: 897-910.
- [11] M. Bruntink, A. van Deursen. Predicting class testability using object-oriented metrics. *SCAM*, 2004: 136-145.
- [12] K. O. Elish, M. Alshayeb. Using software quality attributes to classify refactoring to patterns. *Journal of Software*, 7(2), 2012: 408-419.
- [13] Y. Zhou, Y. Yang, B. Xu, H. Leung, X. Zhou. Source code size estimation approaches for object-oriented systems from UML class diagrams: A comparative study. *Information & Software Technology*, 56(2), 2014: 220-237.
- [14] L. Prechelt. An empirical comparison of seven programming languages. *IEEE Computer*, 33(10), 2000: 23-29.
- [15] T. Gorschek, E. Tempero, L. Angelis. A large-scale empirical study of practitioners' use of object-oriented concepts. *ICSE*, 2010: 115-124.
- [16] B. Ray, D. Posnett, V. Filkov, P. T. Devanbu. A large scale study of programming languages and code quality in Github. *FSE*, 2014: 155-165.
- [17] L. A. Meyerovich, A. Rabkin. Empirical analysis of programming language adoption. *OOPSLA*, 2013: 1-18.
- [18] E. Arisholm, L. Briand, B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1), 2010: 2-17.
- [19] J. Romano, J. Kromrey, J. Coraggio, J. Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's *d* for evaluating group differences on the NSSE and other surveys? *Annual Meeting of the Florida Association of Institutional Research*, 2006: 1-3.
- [20] SciTools Understand. <https://scitools.com/>.
- [21] The Rosetta Code Repository. http://rosettacode.org/wiki/Rosetta_Code.

TAGGINGSENSE: Method Based On Sensemaking For Object-Oriented Source Code Comprehension

Daniel Schreiber
Post Graduate Program in Informatics
(PPGIA) - Polytechnic School
Pontifícia Universidade Católica do
Paraná – PUCPR
Curitiba, Brazil
xiraba@gmail.com

André Menolli
Computer Science Department
Universidade Estadual do Norte do
Paraná - UENP
Bandeirantes, Brazil
menolli@uenp.edu.br

Sheila Reinehr, Andreia Malucelli
Post Graduate Program in Informatics
(PPGIA) - Polytechnic School
Pontifícia Universidade Católica do Paraná –
PUCPR
Curitiba, Brazil
sheila.reinehr@pucpr.br,
malu@ppgia.pucpr.br

Abstract— All software requires maintenance, either for error correction or for implementing updates. However, maintenance is often complex and expensive, and one of the main problems in the high cost of maintenance is the difficulty of understanding the source code of other authors. Thus, this research presents TaggingSense, a method based on sensemaking that aims to reduce object-oriented source code comprehension time on systems maintenance. Through experimentation, it was possible to observe knowledge extracted from the source code, processing, and sharing, to be positively assisted in the source code maintenance and comprehension process, thus bringing benefits such as reduction time spent, quality, and greater security in the changes made.

Keywords-knowledge; sensemaking; source code maintenance; ontology.

I. INTRODUCTION

Software maintenance is one of the activities that consume substantial resources in software projects. In the mid-1980s, the total cost invested in maintenance and improvement accounted for over 60% of the total cost of software systems [1]. In contrast, in the 2000s, total maintenance cost exceeded more than 90% [2]. Maintenance is inevitable because we must ensure updated and efficient software, and this activity is performed for various reasons, such as changes in requirements, bug fixes, component modifications, software improvement, source code optimization, and efficiency improvement, among others [3].

Among several proposed techniques and processes to improve software maintenance, some studies explore cognitive aspects related to software comprehension. With source code being the main maintenance component, comprehension is the predominant factor for providing effective software maintenance, thus allowing the development of computerized systems [4].

Software comprehension corresponds to activities that people perform in order to understand, conceptualize, and reason about software [5]. It is estimated that developers dedicate an average of 40% to 90% of the maintenance effort to the software comprehension process [6] [7]. One of the possible reasons for difficulty in source code comprehension is the lack of knowledge by people without experience, as well as by programmers from other fields.

One method to build knowledge and make sense of things is through sensemaking. Sensemaking is the process of turning circumstances into situations that can be comprehended explicitly in words, and that serves as a catalyst for actions [8]. Weick [8] considers labeling (assigning explicit names) an essential step in sensemaking.

In maintenance activities, it is in the analysis and comprehension stage that those involved do work to extract knowledge and use it to continue with maintenance. During this activity, the acquired knowledge is conserved in people's memories, and such knowledge is divided into two classes: syntactic and semantic [9]. Both semantic and syntactic knowledge are directly and indirectly related to source code comprehension. Many studies and models of comprehension identified different types of knowledge, including knowledge of programming, knowledge of real-world situations addressed by software, and knowledge of the application domain [10].

After comprehension, the coding activity, a process through which developers declare their intentions for the computer, is performed. This activity implies high processing power and storage in the memory of people, because, in addition to the domain, developers need to visualize the organization of objects and routines, as well as the data flow [11]. These challenges, coupled with the effort applied to maintenance and the absence of an ideal solution to these problems, led to the development of this research. It is believed that a comprehension method applied to the source code related to the extraction and dissemination of knowledge can assist in the comprehension process, thus reducing uncertainties and the time dedicated to maintenance tasks.

Therefore, this study aims to develop a method based in sensemaking to reduce object-oriented source code comprehension time on system maintenance. More specifically, it is intended to answer the following question: Is it possible to reduce the time and effort of source code comprehension, and thus increase the quality and efficiency of software maintenance?

II. RELATED LITERATURE

Of all the activities involved in the process of maintenance, comprehension is the most important, as it is considered to be the essential basis for modifying a software product [12]. Studies

show that efforts applied on maintenance are mainly targeted to the comprehension part [11].

Several works were developed related to software maintenance and comprehension, not all of which are focused on serving the same purpose. However, these studies use similar techniques for working on the source code. For example, in research [10], the complexity of understanding a program at the time of maintenance was studied for the purpose of calculations and estimates of effort metrics. Work [13] identified two levels of comprehension: syntactic and semantic. The work proposed in [14], by means of cataloging source code, already seeks to discover programmers' knowledge on application domain. [12] explored a method for maintaining software engineering artifacts "connected" through semantic connections, starting from the source code, by means of ontologies. Work [15] proposes a union of the ontology of code knowledge with domain knowledge, and lastly, work [16] developed source code and documentation ontology to assist in the comprehension process through complex searches inferred on ontology populated from text mining applied in the source code.

The use of ontologies has been significantly explored in software maintenance activities for much of the works highlighted here. Among the techniques for applying ontology to the source code, this paper proposes a new approach: using ontology as a consequence of the knowledge extracted from the source code by using the sensemaking technique. Based on sensemaking, we propose the development and implementation of a method with the principle of formalizing and implementing a folksonomy within the source code, so that it is possible to extract knowledge and maintain it in a knowledge base, with the goal of extracting and disseminating both domain knowledge and the features contained in the source code.

III. TAGGINGSENSE METHOD

In this section, we present the "TaggingSense" method, which supports the steps and the intrinsic processes involved in source code comprehension during software maintenance. This method combines the tagging concepts of folksonomy, and the stages and processes identified by sensemaking, with the goal of accelerating and improving the comprehension process of unknown source codes.

A. Method Structure

From the eight stages of sensemaking (Organizes Flux, Noticing and Bracketing, Labeling, Retrospective, Presumption, Social and Systemic, Action, and Organizing through Communication) conceived by [8], four activities have been defined for the proposed method, and are described as follows:

- **Observation:** consists of the superficial analysis that a programmer performs when starting the maintenance activity. Owing to such observations, in this activity, the programmer formulates ideas and structures based on the experience of past projects.
- **Extraction:** activity related to the extraction and development of knowledge contained in the source code. This activity starts sensemaking. Knowledge is formalized and archived by the programmer with the source code.

- **Organization:** organizing and structuring the extracted knowledge. This activity consists of provided support and the support or rejection of raised ideas and hypotheses in order to improve knowledge structuring. It is in this activity that the programmer identifies phenomena and observed patterns, improves externalization, and catalogs the acquired knowledge.
- **Collaboration:** the main component of this activity is communication. In this activity, the sharing and development of knowledge with the group of people involved in the process occurs through the exchange of experience and the refinement of learning.

Based on the activities defined, details of the method and the steps involved in each activity are described in Table I.

In total, four activities were created with a subtotal of 15 interrelated steps. Each activity has a purpose that serves as input to generate a specific output. The outputs generated by the activities are: (i) formulation and structuring of ideas and hypotheses (observation activity): ideas are formulated and structured tacitly, where externalization occurs in the execution of the next stage; (ii) formalized knowledge (extraction activity): transformation of tacit knowledge into explicit; (iii) restructured and organized knowledge (organization activity): this activity organizes knowledge in a structured way, and enriches existing knowledge with more information; and (iv) knowledge base (collaboration activity): this is the location where all knowledge extracted from the source by one or more programmers is stored.

B. Knowledge Representation

The TaggingSense method proposes the use of a folksonomy-based ontology for organizing and managing tags. In [27], the authors defined folksonomy as the result of a personal free marking (tag) of information and objects for retrieval. The use of tags through folksonomy fits best to factor a demonstration of human thought, compared with those methods related to automatic extraction of text [17]. Through a manual process, the user develops source code sensemaking and identifies a topic/knowledge through tagging. In this process, folksonomy is the result of the sensemaking process designed by the user. One of the strengths of folksonomy is the free assignment of words to features. Annotating a feature with multiple keywords requires less cognitive effort than selecting a single category [18].

Folksonomy is represented through ontology, which serves as basis for supporting the processes. This helps to solve the main problems of folksonomy, such as synonyms, ambiguities, and searches. The main ontologies developed to support the tagging process were evaluated, such as Newman [19], SCOT [20], MOAT [21], Knerr [22], and NAO [23]. After analyzing the available ontologies, the ontology of Knerr [22] was chosen, due to its better compliance with the requirements of the problem, its availability, and easy access to documentation of their classes and properties.

TABLE I. TAGGINGSense METHOD ACTIVITIES AND STEPS

Activities	Steps	Description
Observation	Structure analysis	Preliminary study of the structuring of the source code.
	Technical knowledge and domain search	Improvement of technical knowledge in relation to the source code structure, such as programming language, paradigms, architecture, and standards, in addition to complementary studies related to the domain.
Extraction	Knowledge extraction	Development of domain concepts. Assimilation between domain issues and technical issues related to the source code.
	Tagging	Marking source code through tags. Use of folksonomy to assist, support, and organize tags created during knowledge extraction.
	Externalization	Knowledge articulation occurs, i.e., transformation of tacit knowledge into explicit or usable knowledge. This task represents the continued task of Tagging.
	Guides	Improvement of source code tagging. Tagging is structured in a way that helps programmers find such markings in the source code through waypoints.
	Enrichment tags	Development of new concepts related to those already developed and identified by means of tags.
Organization	Knowledge refinement	Refinement of points related to application domain. Revaluation and continuation of “Knowledge extraction” task of previous activity.
	Tag re-evaluation	Importance validation with project and domain. Redundant tags are eliminated; common tags are reused in the project.
	Cataloging standardization	Standardization between the terms already created.
	Release	All new created tags have visibility property set to private because they are developed at this stage and can change or be eliminated by the creator.
Collaboration	Storage	Throughout the process, extracted knowledge is stored in a database called knowledge base, through ontologies meaning.
	Sharing	Database must be shared with everyone specifically involved in the process of project maintenance.
	Refinement	Enhancement and improvements in tags created by other programmers.
	Reuse	Reuse of tags created by other programmers.

IV. TAGGINGSense ENVIRONMENT

To support the TaggingSense method, we implemented an environment to allow tagging the source code in order to assist in its comprehension. The tagging process consists of manually extracting source code knowledge, and adding it in the folksonomy ontology. This information corresponds to keywords for the tag, date, time, and creator, in addition to the class, method, variable, or related code snippet, that can be

inserted to the same tag created for other individuals. This environment was built as a plug-in for the Eclipse development IDE (integrated development environment). In this environment, interaction starts from the programmer’s comprehension of the source code from the bottom to the top (“bottom-up”) of the source code lines that represent the domain knowledge, through the identification of relevant chunks. Chunks are code portions that programmers can recognize. Large chunks contain several smaller chunks [16].

After this step, it is necessary for the source code to be processed and synchronized with the source code ontology. In the environment, SCRO (Source-code Ontology) is used as the source code ontology because it was created to support the main tasks of software comprehension through the explicit representation of conceptual knowledge found in the source code [24]. This synchronization consists of the extraction of information from the project’s source code, such as methods, input and output values of each method, attributes, and classes, and population of the source code ontology.

Once the source code ontology is populated, the next step is to interact with the folksonomy ontology. This allows new individuals created in this ontology through the creation of tags by the programmer to be associated according to the instances of individuals of the source code ontology. Lastly, the process results in a knowledge base that contains all created tags and their respective associations, derived from the domain knowledge received from the source code. The knowledge base consists of the very folksonomy ontology populated and inferred by inference mechanisms. The environment implementation is presented in the next subsection.

A. Environment Implementation

The environment was implemented according to the assumptions of sensemaking, folksonomy, and knowledge base. In addition, six implementation requirements were raised to support the source code comprehension process. They are:

- Requirement 1: query and record domain information in the folksonomy ontology. Information refers to the knowledge acquired during the comprehension process, and should be semantically linked to allow queries and inferences (reasoning).
- Requirement 2: populating source code ontology. The plug-in must provide a method for extracting semantic information from the source code and automatically populating the source code ontology.
- Requirement 3: populating folksonomy ontology. Populating the domain ontology, which corresponds to the tags created, must be performed manually. As a result of sensemaking, the source code comprehension process is best developed manually because it is at this moment that the user assimilates and understands the source code.
- Requirement 4: searches of instances in the ontology.
- Requirement 5: allows to create, connect, provide, identify, query, and share tags during the source code comprehension process.

- Requirement 6: integration with the working environment.

In order to automatically extract the source code and allow direct interaction with the user, the system was designed and developed based on Eclipse 3.6 and Java 6 platform.

The source code ontology is automatically populated by the plugin, through QDox library [25], whereas the tags manipulation is manual, according to user action. The source code is the only input software artifact, whereas the remaining entries in the system are through manual intervention. Queries by created and populated tags occur through SPARQL-DL with OWL-API support library because there was no native support for SPARQL queries during the development of this research.

Based on the requirements for extraction and manipulation of gathered knowledge, the TaggingSense plug-in was developed to manipulate ontologies and tagging in the source code, with the following functionalities: (i) Display tags related to the selected code: from a window, it is possible to analyze the relationship between the programming-related object and the associated domain concept (tag); (ii) Display tags in tree format: from the list of tags, it is possible to find the source code related to the selected tag; and (iv) Display use of all tags: list all public tags created by any person, in addition to private tags authored by the current user.

In addition to the features described, the plug-in allows the addition of new tags and makes the tags public, thus allowing other users to view the tags and use them collaboratively.

V. EXPERIMENT

To evaluate the feasibility of the method and the environment, an experiment was proposed with the goal of answering the initial question of this research: Is it possible to reduce the time and effort of source code comprehension, and thus increase the quality and efficiency of software maintenance?

To evaluate the experiment, three criteria were defined: (i) programmer behavior: evaluation based on observations from an expert who accompanied the experiment; (ii) development time: this was considered a metric to measure method efficiency; (iii) quality of maintenance performed: an assessment as to whether the requested improvements were implemented as expected.

To conduct the experiment, four IT professionals, who work in a midsize software company, were selected. The selected professionals belong to two distinct classifications: junior, professionals with less than five years of experience in OOP (Object-Oriented Programming), software architectures, design patterns, organization and best coding practices; and senior, programmers with equals or more than five years of experience in system development with knowledge of working on large, complex projects. The participants were requested to make two improvements to an existing system that was unknown to them. The system consisted of a salesforce automation project developed in Java language for mobile devices. Its initial release was designed to run on PALM OS, Windows Mobile, and Android devices. The experiment was divided into three parts, each part containing a specific purpose and applied to specific

participants, as summarized in Table II. In addition, a maximum execution time for each maintenance task was stipulated.

TABLE II. EXPERIMENT DESCRIPTION

	Participants	Objective	Procedure
Experiment 1	Junior A Senior A	Evaluate understanding difficulty and comprehending source code of other authors.	Same activity for participants with and without experience. Activity consists of making improvements to existing system. For this experiment, features for using tags were not available, only features offered by IDE.
	Evaluation		
	Improvement time and location		
Experiment 2	Junior B Senior B	Evaluate comprehension of source code of other authors that performed tagging.	Same activity for both types of participants. Source code is not tagged, but developers are allowed to add, share, and use tags to assist maintenance process.
	Evaluation		
	Improvement time and location; name and number of new tags created during the process.		
Experiment 3	Junior B Senior B	Evaluate comprehension of project already tagged by someone familiar with the project.	Repeat experiment 1 with project already tagged. Those involved should use tags as guides to reach system critical point, thus performing maintenance at correct location.
	Evaluation		
	Maintenance time and quality; number of new tags created.		

A. Results

Analysis of the results was performed mainly in a qualitatively manner. In this analysis, the purpose of the experiments was considered, and the experiments were designed so that a comparison could be made, as described in Table III.

In experiment 1, senior participant A showed difficulties when attempting to find the location (class/method) that caused the parameter to perform the validation requested for this experiment. However, he was able to perform the experiment successfully in 16 minutes, and executed the maintenance in the expected class and method. Junior participant A could not find the correct location of the maintenance in the stipulated time. Even after being shown the location where the maintenance should be performed, the participant failed to complete the task successfully within the stipulated time because, although the maintenance was performed correctly, the code was not implemented in the expected method.

In experiment 2, junior participant B did not use the plug-in as a support tool and could not find the correct method where the improvement should be implemented. Senior participant B achieved this improvement in 12 minutes, and did not need to receive any type of help or advice. However, neither senior participant B nor junior participant B implemented an improvement on the desired method and class.

TABLE III. EXPERIMENT COMPARISON

Relationship	Evaluation - Objective
Experiment 1 x Experiment 2	Check maintenance performance without the use of tags (experiment 1) and with the use of tags (experiment 2); evaluate performance of maintenance performed between junior programmers, among senior programmers, and between junior and senior programmers.
Experiment 1 x Experiment 3	Analyze performance of maintenance performed by senior programmer without the use of tags and by junior programmer with the use of tags.
Experiment 2 x Experiment 3	Evaluate impact on improvement maintenance when there are no tags; that is, comprehension is initiated without the aid of previously created domain concepts (experiment 2). Evaluate impact on improvement maintenance when tags are identified previously (experiment 3) and are available to assist in the comprehension process.

In experiment 3, participants had access to the tags. Junior participant B started the maintenance using the available tags. Through the tags, the class attribute that had the value that needed to be changed was easily deduced. After the locating task was performed all locations that called the attribute in question were searched by the programmer in the source code. Every item in each code snippet that was located was verified against the related tag. Junior participant B performed the activity in merely eight minutes, without any type of help or support. Compared with senior participant A who ran the same maintenance in experiment 1 without the aid of tags, junior participant B was faster because senior participant A performed the same maintenance in 16 minutes. In turn, senior participant B, who had access to the tags, implemented the proposed improvement in four minutes; half the time displayed by junior participant B. Table IV presents a summary of the maintenance time required by senior and junior programmers.

TABLE IV. COMPARISON BETWEEN TIME OF SAME MAINTENANCE WITH AND WITHOUT TAG

Participant	Without tags	With tags
Junior Group	30 min	8 min
Senior Group	16 min	4 min

VI. DISCUSSION

In experiments 1 and 2, tag features to be used in the comprehension process were not available to programmers. However, for experiment 3, the tags were made available to assist in the comprehension process. From the results, it can be concluded that sensemaking development is influenced heavily by the availability of features. The group of junior programmers who did not use tags required an average of 30 minutes to perform the proposed maintenance. However, through the tags, this time decreased to eight minutes, demonstrating a 74% productivity improvement in performance.

In the same sense, the senior group performed the same maintenance in 12 minutes, whereas by means of tags, this time decreased to four minutes, showing a gain of 75% for this class of developers

In experiment 2, wherein the tags were not available, but the possibility of creating and using them was offered, only the

group of senior participants benefitted. However, the tags created were used as waypoints (identification of locations), and as memorization topics that were extracted from the source code. Thus, the created tags helped in source code navigation, assisting developers to locate code among the many classes and methods, avoiding them to get lost on source code navigation.

In contrast, in the experiment where the tags were already created and available, only the group of juniors added a new tag. The new tag served the same objective as for the other group, that is, as a waypoint.

We can conclude that in unfamiliar environments, extracting source code knowledge is easier for more experienced developers precisely because they have more experience. It was also observed that in environments where knowledge of the code was already present, senior programmers did not process new knowledge, whereas junior programmers were led by the existing tags, and even added a new related tag. The failure to process new knowledge puts in evidence the conclusion of the study by [26], which showed that there is no interest on the part of software engineers to study application domain knowledge when performing specific maintenance, where only knowledge related to software engineering (programming, development environment, and application implementation) are considered. The authors in [26] concluded that developers cultivate past knowledge, and searching for new knowledge is a costly process that is performed only when there is a clear need for the programmer and there is no easier alternative. According to [26], software engineers attempt to understand only what is necessary for a system to solve the current problem, and then tend to forget the details of what they learned.

Senior programmers in experiment 2 showed an average of 60% higher performance in the same experiment performed by the group of junior programmers. In this experiment, only the senior programmers used the feature for extracting knowledge from the source code. This justifies the fact that sensemaking is best developed when there already exist foundations and past experience [8].

However, as already discussed, in an environment where the knowledge contained in the source code is extracted previously by an expert with greater knowledge, and is made available via tags for those with less experience, a significant gain in performance is demonstrated.

Thus, we can conclude that the proposed method for extracting and sharing knowledge of the source code is sufficiently effective for improving overall performance of the development team.

VII. CONCLUSIONS

The software maintenance field is complex, mainly because it is dependent on a source code comprehension process, an activity that involves greater cognitive effort of the people involved. Several studies have been developed to facilitate code comprehension. However, this process can still be improved. Knowledge extracted directly from the source code through sensemaking is rich in important and valuable details that can be applied to source code comprehension. This knowledge can be best utilized when stored by means of ontologies and disseminated to more people using Semantic Web. With this

process, the knowledge can not only be extracted, but also shared with those involved, thus benefitting the entire team. Through the results of our experiments, we demonstrated that the proposed TaggingSense method is viable because we were able to conclude that knowledge extraction, processing, and sharing assists positively in the process of source code maintenance and comprehension, thus obtaining benefits such as reduced time, increased quality, and greater security in the changes made. We also showed that our proposed method can guide programmers to the exact location of the improvement required, thus causing maintenance to not occur in wrong places that could affect the quality of the program or open the possibility for security breaches. Thus, the main issue of this research could be answered: it is possible to reduce the time and effort for source code comprehension during maintenance. However, we plan to extend the study to a larger number of participants. We also intend to evaluate the reaction of programmers with different educational backgrounds, as well as evaluate the question of the impact of personal and organizational culture and customs.

REFERENCES

- [1] M. Zelkowitz, A. Shaw, and J. Gannon, Principles Of Software Engineering And Design. Prentice Hall Inc., , 1979, pp 157 – 178.
- [2] L.Erlikh, "Leveraging Legacy System Dollars For E-Business," It Professional, Ieee, vol. 2, n. 3, 2000, pp. 17 – 23.
- [3] B. B. Argawal, and S. P. Tayal, Software Engineering. Laxmi Publications: New Delhi, 2007.
- [4] A. Mayrhauser, and A. Vans, "Program Comprehension During Software Maintenance And Evolution," Ieee Computer vol. 28, 1995, pp. 44 – 55.
- [5] W. Meng, J. Rilling, Y. Zhang, R. Witte, S. Mudur, and P. Charland, "A Context-Driven Software Comprehension Process Model," Ieee Software Evolvability Workshop, 2006.
- [6] A. De Lucia, A. R. Fasolino, and M. Munro, "Understanding Function Behaviours Through Program Slicing," In 4th Ieee Workshop On Program Comprehension, Ieee, 1996, pp. 9 – 18.
- [7] A. Telea, and V. Lucian, "Visual Software Analytics For The Build Optimization Of Large-Scale Software Systems," Computational Statistics, vol 26, n. 4, 2011, pp. 635 – 654.
- [8] K. E. Weick, K. M. Sutcliffe, and D. Obstfeld, "Organizing And The Process Of Sensemaking," Organization Science, 2005, pp. 409 – 421.
- [9] B. Shneiderman, Designing The User Interface: Effective Strategies For Effective Human-Computer Interaction, 2nd ed. Addison Wesley, 1992.
- [10] J. Yang, D. Hendrix, K. Chang, and D. Umphress, "An Empirical Validation Of Complexity Profile Graph," In Proceedings Of The 43rd Annual Southeast Regional Conference, Acm, vol 1, 2005, pp. 143 – 149, 2005.
- [11] C. L. Corritore, and S. Wiedenbeck, "Mental Representations Of Expert Procedural And Object-Oriented Programmers," In A Software Maintenance Task. In International Journal Of Human-Computer Studies, vol 50, n. 1, 1998, pp. 61 – 83.
- [12] R. Witte, Y. Zhang, and J. Rilling, "Empowering Software Maintainers With Semantic Web Technologies," Springer-Verlag Berlin, n. 4519, 2007, pp. 37 – 52.
- [13] C. Dasgupta, "That Is Not My Program Investigating The Relation Between Program Comprehension And Program Authorship," In Acm Se '10 Proceedings Of The 48th Annual Southeast Regional Conference, Acm, n. 103, 2010.
- [14] R. Sindhgatta, "Identifying Domain Expertise Of Developers From Source Code," In Proceeding Of The 14th Acm Sigkdd International Conference On Knowledge Discovery And Data Mining, Acm, 2008, pp 981-989.
- [15] H. Zhou, F. Chen, and H. Yang, "Developing Application Specific Ontology For Program Comprehension By Combining Domain Ontology With Code Ontology," In Quality Software, 2008. Qsic '08. The Eighth International Conference, Ieee, 2008, pp. 225 – 234.
- [16] Y. Zhang, An Ontology-Based Program Comprehension Model. Doctoral Thesis, University Of Concordia, Canada, 2007.
- [17] H. Al-Khalifa, and H. Davis, "Exploring The Value Of Folksonomies For Creating Semantic Metadata," International Journal On Semantic Web & Information Systems, vol 1., n. 3, 2007, pp. 13 – 39.
- [18] R. Sinha, "Tagging From Personal To Social: Observations And Design Principles," In Tagging Workshop, World Wide Web Int. Conf., 2006.
- [19] R. Newman, Tag Ontology Design. Available On <Http://Www.Holygoat.Co.Uk/Projects/Tags/>. Accessed On 27 March 2012.
- [20] H. L. Kim, A. Passant, J. G. Breslin, S. Scerri, and S. Decker, "Review And Alignment Of Tag Ontologies For Semantically-Linked Data In Collaborative Tagging Spaces," In Proceeding Of The 2nd International Conference On Semantic Computing, Ieee, 2008. pp. 315 – 322.
- [21] A. Passant, "Using Ontologies To Strengthen Folksonomies And Enrich Information Retrieval In Weblogs: Theoretical Background And Corporate Use-Case," In International Conference On Weblogs And Social Media, Boulder, United States, 2007.
- [22] T. Knerr, "Tagging Ontology—Towards A Common Ontology For Folksonomies," Available On <Http://Tagont.Googlecode.Com/files/Tagontpaper.Pdf>. Accessed On 27 March 2012.
- [23] S. Scerri, M. Sintek, and L. Van Elst, "Handshuch, S. Nepomuk Annotation Ontology (Nao)," Available On <Http://Www.Semanticdesktop.Org/Ontologies/Nao/>. Accessed On 28 March 2012.
- [24] A. Alnusair, "Scro – Source-Code Ontology," Available On <Http://Www.Indiana.Edu/~Awny/Index.Php/Research/Projects-Tools/15-Research/Ontologies/10/>. Accessed On 22 February 2012.
- [25] Qdox: Qdox Java Parser Extractor. Available On <Http://Qdox.Codehaus.Org/>. Accessed On: 31 May 2012.
- [26] M. R. Ramal, R. M. Meneses, and N. A. Anquetil, "Disturbing Result On The Knowledge Used During Software Maintenance," In 9th Working Conference On Reverse Engineering, Ieee, 2002, pp. 277 – 286.
- [27] V. Wal, "Explaining And Showing Broad And Narrow Folksonomies," Available On <Http://Www.Personalinfocloud.Com/2005/02/Explaining_And_.Html>. Accessed On 10 November 2011.

Facilitating Peer Learning and Knowledge Sharing in STEM Courses via Pattern Based Graph Visualization

Emilio Zegarra, Shi-Kuo Chang, Jingtao Wang

Department of Computer Science
University of Pittsburgh
Pittsburgh, PA United States
{ezegarra, chang, jingtao}@cs.pitt.edu

Abstract—High quality education in Science, Technology, Engineering and Math (STEM) majors expects not only the acquisition of comprehensive domain knowledge, but also the mastery of skills to solve open ended and even ill-defined problems. Problem-based Learning (PBL) is usually adopted to achieve such goals. However, PBL requires sustained and in-depth faculty involvement, hence making PBL not scalable. Also, existing knowledge discovery techniques do not facilitate the capture and reuse of solutions to recurring problems. To address these challenges, we present MicroBrowser, an interactive Q&A system augmented with 2D discussion visualizations and pattern based expertise sharing interfaces. MicroBrowser allows learners to browse and navigate important discussions in PBL based on topic similarity encoded by node proximity in a knowledge graph. MicroBrowser also provides a set of pattern based expertise sharing interfaces to allow both learners and instructors to highlight, share and reuse major findings in PBL. Through a 32-subject study, we found MicroBrowser to be effective at facilitating knowledge discovery. Moreover, students understood and were able to use design patterns to complete open ended tasks.

Education, knowledge, sharing, discovery, pattern, graph, visualization

I. INTRODUCTION

Work on improving education to students in STEM majors has allowed students to acquire important domain knowledge. Yet, upon graduation, students are faced with open ended and ill-defined problems for which they might not be prepared. Innovative techniques such as PBL aim to address these limitations by encouraging students to learn by addressing everyday problems [13].

An important aspect of PBL is the in-depth participation of faculty advisors [16]. However, for large faculty-to-student ratios it becomes difficult for faculty advisors to get involved with all students resulting in a reduced time and effort involvement and a longer feedback loop. A technique that has been used in the context of online education via Massive Open Online Courses (MOOCs) to address this challenge is the use of discussion forums and Q&A systems [5][12]. These systems have been found to be a common source of knowledge for students when completing homework tasks [18][20] and for interacting with instructors [12]. While these systems can address the scalability problem of faculty involvement, they introduce new problems. First of all, as knowledge bases grow in size, the sheer size of the accumulated knowledge makes it

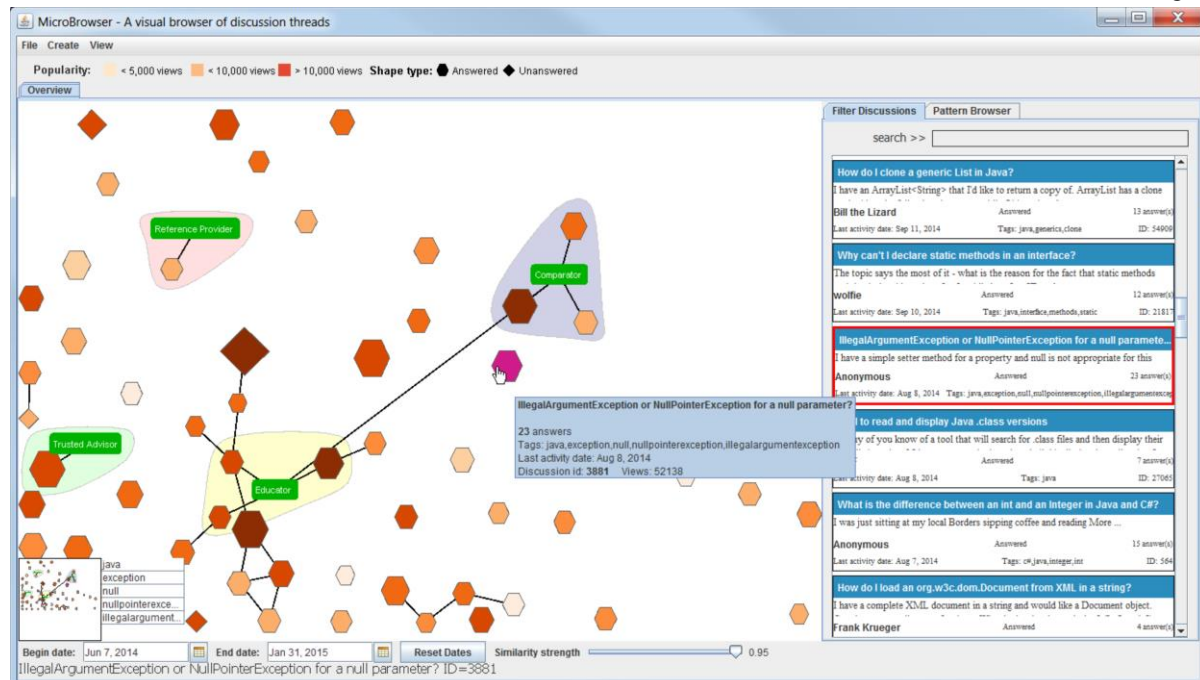


Figure 1. The primary UI of MicroBrowser displaying discussion threads and design patterns. MicroBrowser consists of three components: a knowledge exploration graph, a discussion listing view and a knowledge timeline

harder to locate the desired information. Secondly, existing knowledge discovery techniques do not provide effective means for the capture and reuse of solutions to recurring problems.

To address these challenges, we designed MicroBrowser (Fig. 1). MicroBrowser is a pattern based visualization system designed for educational settings that aims to improve peer-learning by facilitating knowledge discovery and reuse. Its contributions are the ability to discover knowledge from large online discussion boards by using visualization techniques as well as the introduction of design patterns as a method to not only discover knowledge but also to improve peer-learning by facilitating knowledge creation and reuse. In a 32-subject user study, we found that MicroBrowser facilitates the completion of knowledge discovery and reuse tasks when compared to traditional Q&A systems. We also found that for open ended tasks, students made use of design patterns to complete the tasks in over 50% of the time. Also, results from our subjective evaluation indicate students understood the concept of design patterns and found the features in MicroBrowser to be effective and useful. Finally, students found MicroBrowser to be easy to use.

II. RELATED WORK

A. Visualizing Discussion Forums

Discovering of information thru visualization is not limited to finding relevance between posts, but also about finding structure[19]. Several approaches have been used to address forum visualization. ForAvis[23] uses the “overview, zoom and filter” visualization technique as well as color coding to provide a layered approach to displaying information. In [7], the authors found that using treemaps, with color and size encoding, made finding largest and most active discussions faster than traditional text based interfaces. VIDI Toolbar [21] displayed Political Science discussion on a graph with node proximity encoding similarity and cluster of nodes representing topics. Anagora[10] visualized discussion activity along a horizontal chronogram axis. The length of the discussions represented activity and high activity in a forum was represented thru discussion overlap.

In MicroBrowser, we learned from these works on discussion forums and extended them to a Q&A system in an educational domain.

B. Using Design Pattern

Design patterns are a shared language used to communicate proven solutions to recurring problems. Christopher Alexander [1] crafted the notion of patterns to facilitate design and construction of towns and buildings.

The adaptability benefits of design patterns have allowed their extension to other contexts. Gamma et.al used patterns to document recurring problems and their solutions in object oriented programming[8]. Pedagogical patterns assist instructors in preparing effecting instructional material based on learned experience from other instructors[3][14]. Also, human computer interaction and user interface design patterns have been defined to assist with valuable UI design solutions [15][22].

MicroBrowser takes advantages of the benefits of design patterns and applies them to discussion threads in an educational domain. Different from pedagogical patterns, design patterns in MicroBrowser are intended to address the problem of answering recurring questions posted in Q&A systems. MicroBrowser also provides pattern browsing capabilities which allow students to discover pattern patterns and reuse them by either associating them to discussion threads or using their solutions to answer similar questions.

III. THE DESIGN OF MICROBROWSER

The design goals of MicroBrowser are to: a) Identify, create and reuse design patterns in discussions and b) Make it easier to visualize and browse discussion threads and associated patterns.

MicroBrowser was implemented on Java using the Prefuse[11] library.

A. Data Processing and Graph Generation

Topic Modeling facilitates the analysis of large volumes of unstructured text. We use the topic models information to quantify similarities between discussions. For our corpus, we created a text document for each discussion by combining the question title, question text, tags and the answers’ text. Stop words were removed from the documents. We used the Mallet [17] topic modeling toolkit to perform topic analysis and build our topic model. The initial model parameters were set to the default values except for the maximum number of topics which was set to 100 and the maximum iteration was set to 2000. Using the topic model, we created a dissimilarity matrix between each discussion. We first used (1) to compute the similarity between two discussions and (2) for their dissimilarity and to populate our dissimilarity matrix.

$$similarity(i, j) = \frac{similar_topics[i][j]}{count_topics[i]} \quad (1)$$

$$dissimilarity(i, j) = 1 - similarity(i, j) \quad (2)$$

As in [21], we then applied Multi-Dimensional Scaling (MDS) to the dissimilarity matrix to reduce the discussions to a 2-dimension space for plotting. We stored the resulting similarity value between discussions separately to be used when retrieving related discussions. Finally, we used the GraphML¹ format to describe the structural properties of the knowledge base with nodes representing discussions and edges representing relationship. For each discussion, we created a connection to each of its top 20 most similar discussions. For the case of design patterns, we build a connection to each associated discussion. For each edge created between two nodes, we stored the calculated similarity value, except for design pattern edges where we set the value to 1.

¹ <http://graphml.graphdrawing.org>

B. Knowledge Graph View

Similar to [23], we displayed discussions in an overview graph. Thru visualization techniques, students can use their visual abilities to discover relationships, structures or patterns in discussion threads.

We used the shape of a discussion node to encode whether a discussion was either answered or unanswered, with answered ones displayed as a hexagon and unanswered ones displayed as diamond. Having the name of the pattern available makes it easier to identify them. Thus, we used a green label node to display design patterns. To highlight associations to design patterns, we build a convey circle around those discussion nodes belonging to the design pattern and centered the design pattern node among them.

We encoded popularity using a gradient color scheme similar to [2][23] with darker shades denoting more popular items.

A connection between a discussion node and a pattern node indicates that a discussion node is an instance of the pattern. A connection between two discussion nodes indicates that the two discussions are related to each other. A challenge we faced with the number of connections between nodes was that it was cluttering the drawing area making it unreadable. We address this by using progressive disclosure [4] and used the similarity strength value assigned to each connection or edge between two discussions to determine the color intensity of the similarity with more similar items darker. We also added a similarity slider to allow students to control the visibility of the nodes based on the similarity strength.

C. Discussion Listing View

Traditional text based browsing schemes provide simple and effective means for displaying textual information that cannot be represented in a graph node. The discussion listing view provides a text based browsing scheme to allow students to access more information about the discussions that would have been difficult to include in the graph. Since we provided two

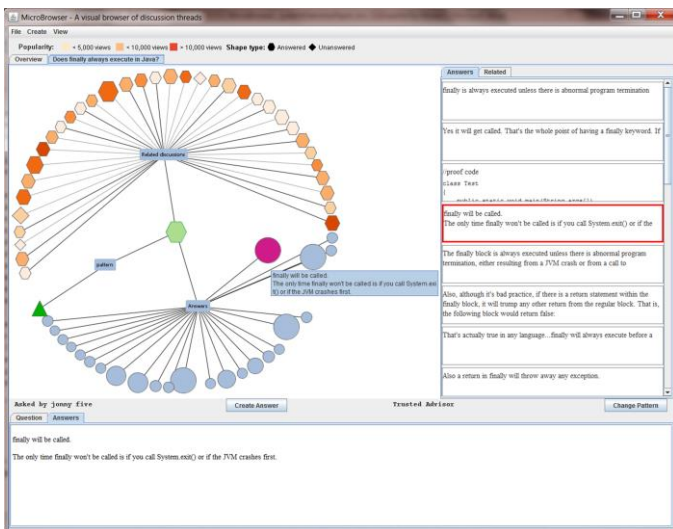


Figure 2. Discussion details view providing question details, answers, related discussions and design patterns. In this figure, a student has selected an answer. The selected answer is brought into focus on the answers list to the right and its content is displayed in the *Answers* tab below.

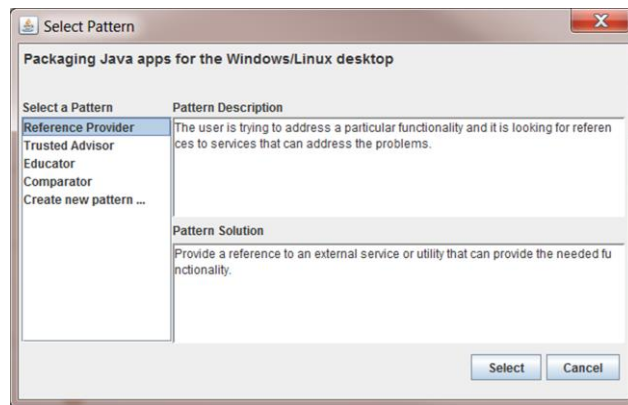


Figure 3. Pattern selection dialog allows students to select existing or create new design patterns.

different browsing approaches (graph and text-based), it was important to ensure proper integration between the two. We enabled bi-directional synchronization between the graph and the listing view. As students moved the mouse over a node in the graph, the corresponding entry in the discussion listing view was brought into focus. Likewise, if a student was more comfortable browsing discussions in textual form, as they moved the mouse over the discussions the corresponding node in the graph got highlighted.

The discussion view also provides keyword filtering to allow users to filter out results.

D. Knowledge Timeline

Only providing keyword filtering might not be enough. In [10][9], the authors allowed users to browse discussions thru their temporal attributes. In an educational setting, time information is very important as discussions can be associated around quizzes or deadlines. To support such scenarios, the knowledge timeline allows students to filter discussions by specifying starting and ending dates. With the timeline in combination with the keyword filters, students can further narrow the available discussions.

E. Discussion Details View

For our discussions, we wanted to represent an organization of answers, related discussions and associated design patterns. In [2], the authors found that using a tree layout provides a simple way to represent perceptual organization. We extended the approach and built a hierarchy for answers, related discussions and design patterns (Fig. 2). The discussion detail view extends the color and size encodings from the knowledge graph view. Textual content for the question and answers was available from the corresponding tab.

F. Associating Design Patterns

When students open up the pattern selection dialog, Fig. 3, they are presented with a list of available patterns they can select from. Selecting a pattern displays its description and the solution. The description helps the student to identify the appropriate question type being evaluated. The solution provides the student with a recommendation for how to answer the type of question.

Once a pattern is selected, the discussion becomes an instance of that pattern. If students need to find samples of how similar questions were answered, they can look at other instances of discussions associated to the selected pattern.

Finally, [6] found that tools that manage pattern collections should be able to store and organize the patterns for easy access and exploration. The Pattern Browser view provides such functionality and it is available from the main navigation window along with the discussion listing view.

IV. USER STUDY

We conducted a 32-subject user study of MicroBrowser to find out:

- Does MicroBrowser promote peer-learning and knowledge discovery? Were students able to discover and reuse knowledge efficiently?
- Did students understand the concept of design patterns? Were design patterns helpful for completing knowledge discovery and reuse?
- Were the design features of color coding, node sizes, keyword and timeline search and discussion overview effective and easy to understand?
- Is MicroBrowser easy to use?

A. Experimental design

To validate the effectiveness of MicroBrowser (MB), we compared it against a state-of-the-art Q&A system (Q&A). Subjects were randomly assigned to start with either MB or Q&A. The user study consisted of three parts. First, students completed a brief tutorial and were asked to complete a total of 12 tasks. After each task, students rated the perceived difficulty of completing the task. Then students completed 12 similar tasks using the other system. There was no time limit for completing these tasks. Finally, students completed an exit questionnaire.

B. Task Descriptions

We defined 12 tasks around 4 scenarios:

1) Discovering knowledge using keywords

Task 1 asked students to find a question about a given keyword. Task 2 asked students to find the most popular discussion for a given keyword. Task 3 asked students to first find an unanswered question about a given keyword and then find related discussion suggested by the system.

2) Discovering knowledge using timeline

Task 4 asked students to find any discussion that occurred during a given timeframe. Task 5 asked students to find discussions about given keyword that occurred during a given timeframe. Task 6 asked students to find the most recent unanswered question.

3) Using Patterns for knowledge reuse

Task 7 asked students to find examples of discussions whose answer either provide recommendations or advise or suggested the use of references. Task 8 asked students to modify a given discussion such that other students looking for similar approaches to answering the question could reference it. Task

9 asked students to submit a given answer to an unanswered discussion. Then they were asked to modify the discussion such that other students could reuse how the answer to the question was presented.

4) Using Patterns for knowledge discovery

Task 10 asked students to find discussions based on the approach taken to answer them. Task 11 asked students to refer to a particular discussion and then find other discussions whose approach to answering it was similar. Task 12 asked students how they can find recommended ways to answer discussions.

C. Participants and Apparatus

We recruited 32 subjects (6 female) over 18 years (18-21: 38%, 22-25: 41%, 26+: 22%) among STEM major students from a local university. Each session lasted 1 hour 30 minutes and participants were compensated with a \$10 Amazon.com gift card.

Participants used a Dell Optiplex 745 (Intel Core2 Duo T6300 1.86GHz, 2GM RAM) using Windows 8 and a 19" display at a screen resolution of 1280x1024. MB ran on top of Eclipse V4.2.2 and Java SE 7u51.

As our baseline Q&A system, we configured an instance of the open source, PHP based Question2Answer² platform and access it via Internet Explorer V10.

D. Data

We loaded the system with the 500 most recent discussion threads and their answers retrieved from StackOverflow³ associated to the tag 'Java'. This data consisted of 4041 records (500 questions, 3541 answers) from 2091 different users. We created 4 initial design patterns and assigned them sample discussions.

E. Usage Behavior

Table 1 summarizes the activities participants performed the most while completing the tasks using MB. We noted that participants visited a median of 182 nodes using the knowledge graph but only 12 nodes using the discussion listing view suggesting a preference for browsing knowledge using the knowledge graph over the discussion listing view. Participants opened an average of 15 discussions and 8 design patterns suggesting that integrating discussion details in the knowledge graph reduced the need to open discussions for information. When students opened the details of a

TABLE 1. DESCRIPTIVE STATISTICS SUMMARIZING PARTICIPANTS' ACTIVITIES USING MICROBROWSER (SD=STD DEV).

Activities	Median	Mean	SD
Open discussion details	15.0	15.37	7.66
Open design pattern details	8.0	8.12	3.88
Open answer details	56.5	81.50	76.49
Perform keyword searches	13.0	13.81	5.14
Visit node on knowledge graph	179.0	182.56	72.20
Visit node on discussion listing	8.0	12.28	13.77

² <http://www.question2answer.org>

³ <http://www.stackoverflow.com>

discussion, they could explore related discussions and answers associated to the discussion. Data shows that on average participants viewed the details of about 50 answers (SD = 76.49) suggesting that MB facilitated knowledge exploration not only of discussion but also of their answers. Finally, participants performed an average of 13 search events.

F. Perception of Difficulty

1) Discovering knowledge using keywords

Results show no statistically significant differences between Q&A and MB for Task 1 ($p = 0.8514$). For Tasks 2 and 3, results show statistically significant differences between the two methods. Data shows that MB is easier to use for completing those tasks (Table 2).

2) Discovering knowledge using timeline

For Tasks 4-6 results show statistically significant differences between Q&A and MB. Data show Task 4 and 5 were easier to complete using MB while for Task 6 Q&A was easier (Table 2).

TABLE 2. STUDENTS' PERCEPTION OF DIFFICULTY FOR TASKS 1-6. SCALE OF 1-EASIER TO 5-MORE DIFFICULT. (GRAY BACKGROUND = $P < 0.05$, WILCOXON SIGNED-RANK TEST, IQR=INTERQUARTILE RANGE, SD=STD DEV)

Task	Q&A		MB	
	Median (IQR)	Mean (SD)	Median (IQR)	Mean (SD)
1	1 (0)	1.25 (0.56)	1 (0.25)	1.28 (0.52)
2	2 (1.15)	2.51 (1.15)	2 (0.59)	1.68 (0.59)
3	3 (1.09)	3.00 (0.83)	1 (0.83)	1.67 (0.83)
4	3 (1.14)	3.03 (1.14)	1 (0.67)	1.50 (0.67)
5	2 (1.10)	2.25 (1.10)	1 (0.56)	1.46 (0.56)
6	1 (1.00)	1.65 (1.00)	3 (0.98)	2.93 (0.98)

3) Using Patterns for knowledge reuse

No statistically significant differences were found for Task 8 ($p=0.0675$) or Task 9 ($p=0.1164$) between the two systems. However, data shows a trend towards MB being easier to complete the tasks. Results show MB was easier to complete Task 7.

4) Using Patterns for knowledge discovery

We found statistically significant differences between Q&A and MB for Tasks 10-12. Students found using MB to be less difficult to complete these tasks.

TABLE 3. STUDENTS' PERCEPTION OF DIFFICULTY FOR TASKS 7-12. SCALE OF 1-EASIER TO 5-MORE DIFFICULT. (GRAY BACKGROUND = $P < 0.05$, WILCOXON SIGNED-RANK TEST, IQR=INTERQUARTILE RANGE, SD=STD DEV)

Task	Q&A		MB	
	Median (IQR)	Mean (SD)	Median (IQR)	Mean (SD)
7	3 (1)	2.83 (1.00)	2 (2)	2.09 (0.92)
8	3 (2.5)	2.90 (1.37)	2 (1.25)	2.37 (1.23)
9	2.5 (2)	2.81 (1.40)	2 (1)	2.40 (1.04)
10	2 (1)	2.60 (1.16)	2 (2)	2.00 (1.09)
11	2 (1.5)	2.50 (1.29)	2 (1)	1.76 (0.81)
12	3 (1.37)	3.37 (1.37)	2 (1.14)	2.30 (1.14)

G. Qualitative Results

The majority of students found the use of design patterns effective and most importantly easy to learn (Fig. 4). This was an important finding as we wanted to validate students could understand their value. Students were ecstatic about their capability and usefulness as noted by the following comments:

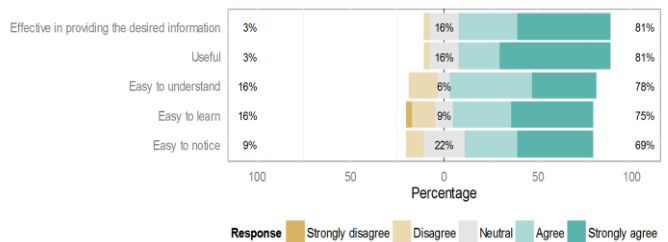


Figure 4. Students' opinion about design patterns. 5-point Likert-scale

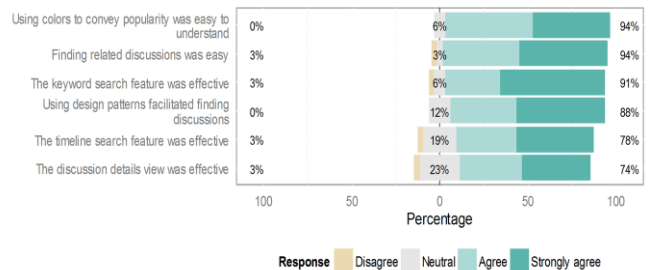


Figure 5. Students' opinion about MicroBrowser features and functionality. 5-point Likert-scale

"Pattern browsing is a great concept", "I really loved the pattern browser. Just a brilliant idea" and "Pattern browser, it is very useful to see "trusted advisor". If I want to see reliable answers and "reference" and "comparator" are very useful."

When directly asked about their feedback, students were very positive on the usefulness and effectiveness of the main features (Fig. 5).

When openly asked to list their most liked features, 56% students cited the integration of design patterns and the pattern browser, 47% the visual representation of discussions, 13% for visualizing relationships, 25% the use of color to encode popularity, 25% for shape encoding, and 22% the knowledge timeline.

Among the features students disliked the most were the initial learning curve (13%), the knowledge timeline (13%), the slow response (9%) and the use of patterns (9%).

Finally, students found MicroBrowser to be easy to use (10-point Likert-scale, median=8).

V. DISCUSSION

In this study, we aimed to determine if our innovative system MB was easy to use and facilitated completion of the tasks when compared to a traditional Q&A. Based on the results, students found MB to be easier to complete most of the tasks when compared to Q&A. Task 1 was very simple which can explain why no statistically significant differences were found between the systems. For Tasks 2 and 3, students benefited in MB from the use of color coding to quickly identify popular discussions and the clear identification of the related discussions in the discussion details view. For Tasks 4 and 5, students benefited from using the knowledge timeline to narrow discussions as well as the ability to filter by both keyword and timeline simultaneously. When using Q&A, students could only sort by recent discussions but had to page thru the discussions until reaching the desired date. Surprisingly, based on results, Q&A was easier for completing

Task 6. Q&A showed unanswered discussions already sorted by update date. This explains why, under Qualitative results, students recommended adding sorting capabilities to MB. Even though Task 7 was open-ended, we wanted to see if students could use the systems to identify instances of reusable knowledge. Using MB, students were able to use the pattern browser and from there refer to the sample discussions. In the case of Q&A, students used different methods, such as performing keyword searches and then reading the details of identified discussions. Sometimes students simply opted for selecting the discussion that matched a keyword search. For Task 8, students benefited by the use of design patterns because 66% of the students used a design pattern to complete the task. For the case of Q&A, students tended to associate a tag to create a classification of the discussions. For Task 9-12, we noted a large percentage of students benefited from design patterns by using them when completing the tasks (50%, 81%, 78% and 77%, respectively). For Q&A, students had to find techniques for identifying recommended discussions such as using keywords or using page view count information. Finally, qualitative results showed students found the MB features to be useful and effective. Students identified some usability issues and also made recommendations for improvements to MB. Among the recommendation were to add sorting capabilities, to improve the knowledge timeline by using a slider, to add filtering by users or question state, to associate more patterns to questions, to add actions on right-click menus and to design for accessibility.

VI. CONCLUSION

In this paper, we presented MicroBrowser, a system that facilitates peer learning and knowledge discovery in classroom settings. Results from our 32-subject user study show reduced difficulty at completing tasks when compared to traditional Q&A system. More importantly, students found benefit in the use of design patterns and found the system and its features effective and easy to use.

The integration of Design Patterns in knowledge discovery and generation is a key innovation in MicroBrowser. Overall, MicroBrowser achieves this with an interactive knowledge visualization and exploration system.

REFERENCES

- [1] Alexander, C. The origins of pattern theory: The future of the theory, and the generation of a living world. *Software, IEEE* 16, 5 (1999), 71–82.
- [2] Cai, Q. and Sheth, N. Visualizing MeSH Dataset using Radial Tree Layout. (2003).
- [3] Carle, A., Clancy, M., and Canny, J. Working with pedagogical patterns in PACT. *ACM SIGCSE Bulletin* 39, 2007, 238.
- [4] Chuang, J., Ramage, D., Manning, C.D., and Heer, J. Interpretation and Trust: Designing Model-Driven Visualizations for Text Analysis. (2012), 443–452.
- [5] Coetzee, D., Fox, A., Hearst, M., and Hartmann, B. Should your MOOC forum use a reputation system? *CSCW'14*, (2014), 1176–1187.
- [6] Deng, J., Kemp, E., and Todd, E.G. Managing UI pattern collections. *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction making CHI natural - CHINZ '05*, (2005), 31–38.
- [7] Engdahl, B., Koksall, M., and Marsden, G. Using treemaps to visualize threaded discussion forums on PDAs. *Proceedings of ACM CHI 2005 Conference on Human Factors in Computing Systems*, (2005), 1355–1358.
- [8] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.
- [9] Gibbs, W.J., Olexa, V., and Bernas, R.S. A Visualization Tool for Managing and Studying Online Communications. 9, (2006), 232–243.
- [10] Giguët, E. and Lucas, N. Creating discussion threads graphs with Anagora. *CSCL '09 Proceedings of the 9th international conference on Computer supported collaborative learning*, (2009), 616–620.
- [11] Heer, J., Card, S., and Landay, J. Prefuse: a toolkit for interactive information visualization. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Press (2005), 421.
- [12] Huang, J., Dasgupta, A., Ghosh, A., Manning, J., and Sanders, M. Superposter behavior in MOOC forums. *Proceedings of the first ACM conference on Learning @ scale conference - L@S '14*, (2014), 117–126.
- [13] Hung, Woei (University of Arizona South, Sierra Vista, A., Jonassen, David H (University of Missouri, Columbia, M., and Liu, Rude (Beijing Normal University, Beijing, C. Problem-Based Learning. *Encyclopedia of the Sciences of Learning*, (2012), 2687–2690.
- [14] Köppe, C. and Utrecht, H. A pattern language for teaching design patterns. *Proceedings of the 18th Conference on Pattern Languages of Programs - PLoP '11, Part 2* (2011), 1–16.
- [15] Lin, J. and Landay, J.A. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, (2008), 13.
- [16] Massie, D. and Massie, C. Framework for organization and control of capstone design/build projects. *Journal of STEM Education* 7, 3 (2006), 36–43.
- [17] McCallum, A.K. MALLETT: A Machine Learning for Language Toolkit. 2002. <http://mallet.cs.umass.edu/>.
- [18] Seaton, D.T., Bergner, Y., Chuang, I., Mitros, P., and Pritchard, D.E. Who does what in a massive open online course? *Communications of the ACM* 57, 2014, 58–65.
- [19] Seo, J., Croft, W.B., and Smith, D. a. Online community search using thread structure. *Proceeding of the 18th ACM conference on Information and knowledge management - CIKM '09*, (2009), 1907.
- [20] Sonnino, E. 4 Ways to Get the Most out of a MOOC. *Learning Advisor*, 2013. <http://blog.studentadvisor.com/four-ways-get-mooc/>.
- [21] Trampuš, M. and Grobelnik, M. Visualization of online discussion forums. *Workshop on Pattern Analysis Applications 11*, (2010), 134–141.
- [22] Vanderdonckt, J. and Simarro, F.M. Generative pattern-based design of user interfaces. *Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems - PEICS '10*, (2010), 12–19.
- [23] Wanner, F., Ramm, T., and Keim, D. ForAVis: explorative user forum analysis. *WIMS '11: Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, (2011).

Scaffolding MATLAB and Octave Software Comprehension Through Visualization

Ivan de M. Lessa, Glauco de F. Carneiro
Universidade Salvador (UNIFACS)
Salvador/Bahia, Brazil
ivan.lessa@gmail.com,
glauco.carneiro@unifacs.br

Miguel P. Monteiro
Universidade Nova de Lisboa (UNL)
NOVA LINCS
Lisbon, Portugal
mtpm@fct.unl.pt

Fernando Brito e Abreu
Instituto Universitário de Lisboa
(ISCTE-IUL)
Lisbon, Portugal
fba@iscte-iul.pt

Abstract— Multiple view interactive environments (MVIEs) provide visual resources to support the comprehension of a specific domain dataset. For any domain, different views can be selected and configured in a real time fashion to be better adjusted to the user needs. This paper focuses on the use of a MVIE called *OctMiner* to support the comprehension of MATLAB and GNU/Octave programs. The authors conducted a case study to characterize the use of *OctMiner* in the context of comprehension activities. Results provide preliminary evidence of the effectiveness of *OctMiner* to support the comprehension of programs written in MATLAB and Octave.

Keywords – software visualization; MATLAB/Octave; software comprehension.

I. INTRODUCTION

Multiple view interactive environments (MVIE) provide resources to support data analyses and unveiling information that otherwise would remain unnoticed [1][4]. This work is focused on MATLAB [8] and Octave [11] programs, following reports in the literature that indicate a lack of support for the comprehension of programs coded in these languages. We contribute to fill this gap by implementing a MVIE named *OctMiner*. Following previous research on this topic [2][9], we conducted a case study using *OctMiner* to support the comprehension of MATLAB/Octave programs, which aims at characterizing the MVIE support to identify crosscutting concerns.

This paper is structured as follows: section II describes key functionalities of *OctMiner* and its architecture; section III presents two case studies to characterize *OctMiner* as a means to support MATLAB/Octave program comprehension; section IV proposes a set of usage strategies to be performed with *OctMiner* for comprehension purposes. Finally, section V presents the final considerations and outlines opportunities for future work.

II. MULTIPLE VIEW INTERACTIVE ENVIRONMENTS

Visualization is a means of providing perceivable cues to several aspects of the data under analysis to reveal patterns and behaviors that would otherwise remain unhighlighted and unnoticed [13]. Card et al. [1] proposed a well-known reference model for information visualization. According to them, the creation of views goes through a sequence of successive steps: pre-processing and data transformations, visual mapping and view creation. Carneiro and Mendonça [3] extended this model

to adapt it to the context of MVIEs. Figure 1 shows the extended model, emphasizing that the visualization process is highly interactive. Moreover, it enables the combined use of resources of a multiple view interactive environment. The process starts with original (raw) data obtained from a repository that undergoes a set of transformations to be organized into data structures suitable for information exploration. This process is called *data transformation* [3]. Next, the data structures are used to assemble visual data structures. Those structures organize data properties and visual information properties in ways that facilitate the construction of visual metaphors. This step defines the mapping from real attributes – which are derived from the data properties, software attributes, in our case – to visual attributes such as shapes, colors and positions on the screen. This process is called *visual mapping* [3]. It is important to highlight that these activities do not deal with rendering, but rather with building suitable data structures from which the views can be easily computed and rendered. The final step, presented in Figure 1, is the *view transformation*, aimed at drawing the information on the screen to produce the views. In this step, a specific visual scene is actually rendered on the computer screen [3].

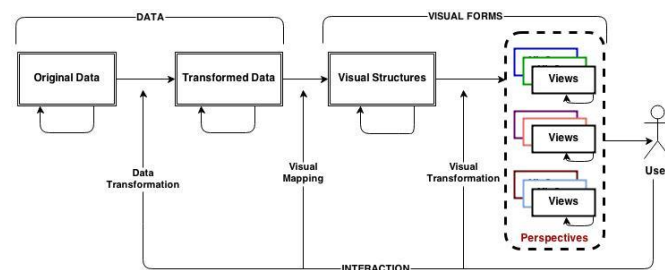


Figure 1. An Extended Reference Model for MVIEs [3]

Nunes et al. [10] proposed a toolkit implemented as a Java Eclipse plugin from which MVIEs could be developed. The plugin provides a basic structure that allows the creation and inclusion of new resources and functionalities to develop MVIEs. Figure 2 presents the way the toolkit was used and extended by other plugins to comprise the SourceMiner MVIE. This MVIE was originally developed to support the comprehension of Java source code. As can be seen in the figure, the extension points of the toolkit.aimv plugin enable the inclusion of new plugins to the MVIE. Each of the extension points conveyed provides an interface with methods and their respective signatures. In the case of *OctMiner*, we needed to access and transform raw data – the Abstract Syntax Tree (AST) of MATLAB/Octave programs – to a format compatible with the

visual data structure. According to the extended reference model for MVIEs, this is a requirement to feed the views.

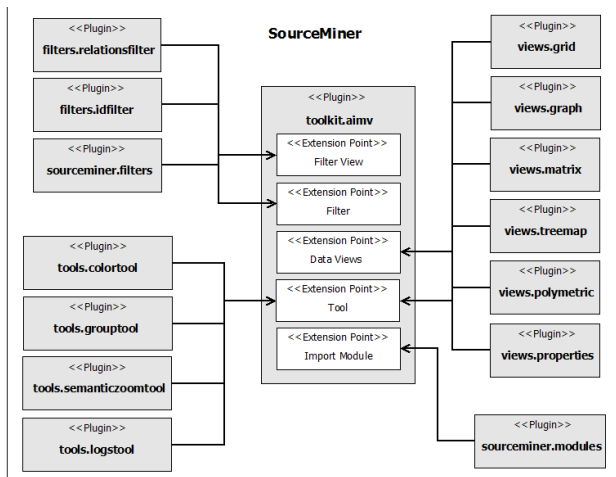


Figure 2. The MVIE SourceMiner [10]

Figure 2 presents a set of plugins that comprise the SourceMiner MVIE. The following guides are available to help MVIE developers: (1) Data Transformation: to extend the plugin Import Module to implement the plugin *sourceminer.modules*; (2) Creating and Applying Filters to extend the plugins Filter and Filter View; (3) Creating Tools to extend the plugin Tools; (4) Creating Views to extend the plugins Data Views and Tools. These guides are available at [14].

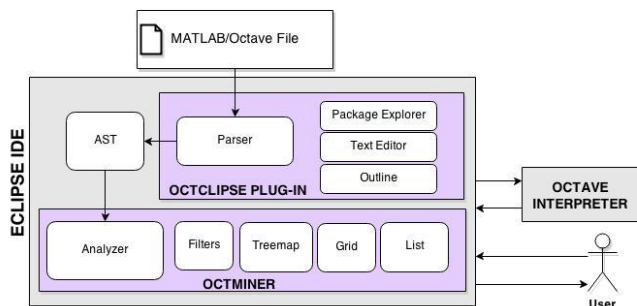


Figure 3. *OctMiner* Architectural Overview [7]

The goal of the toolkit is to provide an infrastructure to develop MVIEs for different domains. The domain targeted in this paper comprises programs written in MATLAB/Octave.

A. THE MATLAB AND OCTAVE PROGRAM LANGUAGES

MATLAB is an interpreted language very popular among students and researchers of physics, biomedical engineering and related areas. It is not uncommon that a young engineer is fluent in using MATLAB, but hardly familiar with C, and even less of Fortran [5][15]. MATLAB has been used to teach linear algebra, numerical analysis, and statistics. Since the MATLAB language is proprietary, a similar language, named Octave was developed, and is distributed under the terms of the GNU General Public License. It was originally conceived in 1988 to be a companion programming language for an undergraduate-level textbook on

chemical reactor design. Due to the similarities between these languages, it is possible to interpret MATLAB programs in the interpreter of the GNU/Octave with no major problems. The main differences among the two languages are as follows: i) Some similar routines can have different names in each language; ii) Comments in MATLAB are written after “%” while in Octave you can use both “%” and “#”; iii) In MATLAB the control blocks (while, if and for) as well as the functions delimiter all finish with “end” while in Octave you can also use “endwhile”, “endif”, “endfor” and “endfunction” respectively; iv) In MATLAB the not equal to operator is “~=” while in Octave “!=” is also valid; v) MATLAB does not accept increment operators such as “++” and “--”, while Octave accepts them.

B. THE AIMV OCTMINER

The main motivation for representing concerns manifested in MATLAB/Octave code in a MVIE is the enhancement of the comprehension activities. The plugin structure supporting the MVIE toolkit is the same as presented in Figure 2. The main difference is that in this case the focus is on MATLAB/Octave rather than Java. Figure 3 depicts the main four elements of *OctMiner*: the Eclipse IDE RAP/RCP (Rich Clients and Rich Ajax Applications), the *Octclipse* plugin, the Octave interpreter and the MVIE toolkit proposed in [10]. The Eclipse IDE enables its extension through the use of plugins. The MVIE toolkit does this to provide its functionalities as well as enabling the tailoring of the MVIE tailoring for the analysis of data from different domains, e.g., the data gathered from MATLAB/Octave programs.

We implemented an Analyzer module as presented in Figure 3, which is analogous to *sourceminer.modules* – see Figure 2. It is an extension of the Import Module, whose goal is to import and convert data from the original data repository to be represented in the multiple views. The *Octclipse* plugin also provides an Octave development environment built on top of Eclipse's Dynamic Languages Toolkit. This environment enables programmers to create Octave scripts (*.m files), edit them in a multi featured text editor, run the Octave interpreter and see results displayed in the IDE's console. *OctMiner* is available at [14].

To provide a short illustration of the visualization capabilities of *OctMiner*, Figure 4 shows a typical visualization scenario. Part A is the Project Explorer, presenting all the repository files; Part B is the Outline, showing the functions and variables of a given file, when it is selected in the Project Explorer. Part C provides editing access to the routine's code. Part F is a filter dashboard. Parts D, E and G are views corresponding to several different visualization metaphors. For instance, the Treemap view (G) provides panoramic view, e.g., of how names of routines are distributed in the file repository. Colours represent different concerns (be they crosscutting or no). We use the term “token” to refer to routine names from the MATLAB/Octave systems. The List view (E) presents a list of the files from the repository. The Grid view (D) is be used to identify the tokens

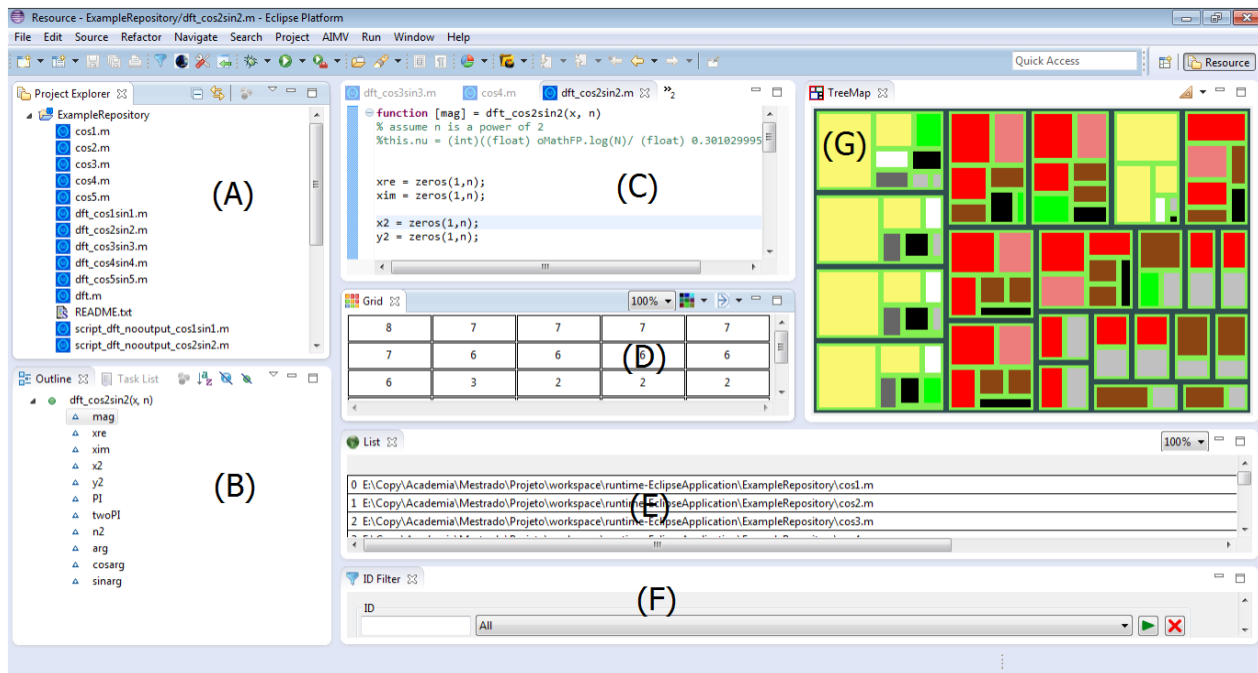


Figure 4. A Typical Scenario of *OctMiner* Use in the Eclipse IDE [7]

used in the repository along with several different metrics, e.g., number of occurrence of each token in each file or in the whole repository. This view can also be presented in several orderings, depending on what is convenient. Full details on the visualizations are provided in our ITNG paper [7].

III. COMPREHENSION ACTIVITIES WITH OCTMINER

This section presents a case study to characterize the use of *OctMiner* in comprehension activities. In it, we investigate the following question: to which extent *OctMiner* provides effective support to identify potential symptoms of crosscutting concerns in MATLAB programs? In the study, we analyze 22 MATLAB image processing routines. The goal is the identification of the dual symptoms of scattering and tangling in the routines, as supported by *OctMiner*. *Scattering* [12] is the degree to which a concern is spread over different modules or other units of decomposition. *Tangling* [16] is the degree to which concerns are intertwined to each other in the same routines. Both scattering and tangling are indicators of the presence of crosscutting concerns in program code.

The case study explores the potential of tokens to be indicators of the scattering and tangling symptoms. The approach is as follows: sets of tokens can be associated to a given concern, which ideally would be modularized into its own file, with no additional concerns. When the concern is not modularized, its code is scattered across multiple files and its associated tokens are found in such files – an indicator of scattering. Often, such files also betray the presence of tokens categorized under multiple concerns – an indicator of tangling.

To explore the aforementioned approach, participants performed the following activities: i) Identify tokens most commonly used in the 22 routines; ii) Characterize the

localization among files of the most commonly used tokens to assess the symptoms of scattering; iii) Characterize the relationship between the most commonly used tokens and other tokens in the files to assess the symptoms of tangling; iv) Determine the category (concern) to which the most commonly used tokens belong; v) Using the category of each token, identify the main functionalities (concerns) of the program. Using this approach, it was possible to identify the top most commonly used tokens in the analyzed routines and that this same tokens presented evidences of scattering. This study was the starting point for the use of *OctMiner* in comprehension activities.

We identified the following limitations in this study: considering that the routines were already analyzed by *OctMiner*, any new modification in the original routines will not be reflected in the views until a new analysis is performed to obtain these modification from the repository. In addition, the user can only select the predefined color in *OctMiner*. It is also not possible to define new colors in this version of *OctMiner*. The need to configure the XML file with the tokens is also a limitation. To overcome it, we intend to provide a XML file with a large number of MATLAB and Octave functions and their respective categories.

We recognize that *OctMiner* may not be able to provide support for all kinds of comprehension needs. To better characterize and validate its range of applicability, we plan additional studies (see section V). Another potential threat to validity is that both design and execution of the study were performed by the same person. To overcome this issue, further independent experiments will be carried out to compare results more thoroughly.

IV. PRELIMINARY STRATEGY BASED ON *OCTMINER*

Results from this case study enable us to propose a preliminary usage strategy based on *OctMiner* for comprehension purposes. The strategy includes a comprehension question as its starting point, which drives subsequent steps. The question is related to tangling and scattering, using a set of tokens from programs of a repository as a basis. Table 3 presents the steps proposed from evidences collected from this case study.

Table 3. A Proposed Set of Usage Strategies

Suggested Steps
1 - Select a question: the programmer needs to identify an issue relevant for his daily activities. Answers to the question should be available considering that the routines used in the code should be registered in the <i>OctMiner</i> configuration file.
2 - Identify a target routine: it should be the routine that plays a relevant role in the code of the primary solution to the selected question.
3 - Locate repositories that use the target routine: since <i>OctMiner</i> aims at assisting the comprehension of a given target routine, it is desirable that routines using the target routine provide good examples and be the subject of analysis.
4 - Identify the routines and their respective categories available in the official documentation: alternative routines used in the repository selected in Item 3 must also be identified. MATLAB and Octave routines are categorized in the official language sites of MATLAB and Octave.
5 - Register the target routine as well as other routines from the repository in the <i>OctMiner</i> configuration file: the routines should be registered in <i>OctMiner</i> configuration file using their specific group, identified according to Item 4.
6 - Create a To-Do list for identification through visualization: activities that the user must perform should be described so that the study is conducted as well as possible within <i>OctMiner</i> .
7 - Implementation of the proposed activities: the user must run <i>OctMiner</i> according to the activities set out in Item 6.
8 - Answer the original question: to prove the effectiveness of the tool, the user should be able to answer the question that started the process in Item 1.

V. CONCLUSIONS AND FUTURE WORK

This paper presents the following contributions: a) the provision of an environment called *OctMiner* for the comprehension of MATLAB/Octave routines supported by multiple views; b) Evidences of the effectiveness of *OctMiner* to support the identification of symptoms of code tangling and code scattering as discussed in the study presented at section III; c) the initial version of a sequence of steps for a strategy for the usage of *OctMiner* for comprehension purposes.

A previous paper by the same authors describing the architecture of *OctMiner* along with an illustrative example of its main functionalities in a real scenario of program comprehension, was presented at ITNG'2015 [7]. An extended version of the present paper, where the validation case studies are described in detail and additional information on the proposal is provided, will appear in the proceedings of ICCSA'2015 in Canada.

We will soon conduct a new version of a more detailed study, based on answers posted at popular question-and-answers sites (e.g., *StackOverflow*). We are planning research questions to assess the extent to which *OctMiner* provides effective support to clarify programmer's issues. We believe *OctMiner* can help programmers in understanding the context of use of a routine through *OctMiner's* visualizations. Our goal is to gather evidence of the effectiveness of *OctMiner* in supporting acquisition of insights by means of the visualization of target routines. We will base the next study on routines referred in posts from question-and-answers sites. The authors would like to thank the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES) for their financial support.

REFERENCES

- [1] Card, S. K., Mackinlay, J. and Shneiderman, B. Readings in Information Visualization Using Vision to Think. San Francisco, CA, Morgan Kaufmann, 1999.
- [2] Cardoso, J.; Fernandes, J.; Monteiro, M.; Carvalho, T.; Nobre, R. Enriching MATLAB with aspect-oriented features for developing embedded systems. *Journal of Systems Architecture* 59 (2013) p. 412-428.
- [3] Carneiro, G.; Mendonça, M.. SourceMiner: Towards an Extensible Multi-perspective Software Visualization Environment. In: Slimane Hammoudi; José Cordeiro; Leszek A. Maciaszek; Joaquim Filipe. (Org.). *Enterprise Information Systems*. 1ed.: Springer International Publishing, 2014, v. 190, p. 242-263.
- [4] Carneiro, G., Silva, M., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., Mendonça, M., 2010. Identifying code smells with multiple concern views. In: XXIV Brazilian Symp. on Software Engineering (SBES 2010), IEEE Comp. Soc., Washington, DC, USA, pp. 128-137.
- [5] Chaves, J.; Nehrbass, J.; Guilfoos, B.; Gardiner, J.; Ahalt, S.; Krishnamurthy, A.; Unpingco, J.; Chalker, A.; Warnock, A.; Samsi, S. Octave and Python: High-Level Scripting Languages Productivity and Performance Evaluation. In Proc. of the HPCMP Users Group Conference (HPCMP-UGC '06).
- [6] Data Explorer - StackExchange. Available at <http://data.stackexchange.com/>.
- [7] Lessa, I.; Carneiro, G.; Monteiro, M.; Abreu, F. A Multiple View Interactive Environment to Support MATLAB and GNU/Octave Program Comprehension. In: International Conference on Information Technology: New Generations (ITNG), 2015, Las Vegas/EUA.
- [8] MATLAB Programming Language. Available at www.mathworks.com/products/matlab.
- [9] Monteiro, M.; Cardoso, J.; Posea, S. Identification and characterization of crosscutting concerns in MATLAB systems. In Conference on Compilers, Programming Languages, Related Technologies and Applications (CoRTA 2010), Braga, Portugal (pp. 9-10).
- [10] Nunes, A.; Carneiro, G.; David, J. Towards the Development of a Framework for Multiple View Interactive Environments. In: International Conference on Information Technology: New Generations (ITNG), 2014, Las Vegas/EUA. p. 23-30.
- [11] Octave Programming Language. Available at www.gnu.org/software/octave/.
- [12] Robillard, M.; Murphy, G. Representing Concerns in Source Code. ACM TOSEM, 2007.
- [13] Spence, R. *Information Visualization: Design for Interaction* (2nd Edition). 2. ed. Prentice Hall, 2007.
- [14] SourceMiner Website. Available at www.sourceminer.org/octminer
- [15] Stenroos, M.; Mäntynen, V.; Nenonen, J. A MATLAB library for solving quasi-static volume conduction problems using the boundary element method. - *Computer methods and programs in biomedicine*, 2007.
- [16] Tarr, P.; Ossher, H.; Harrison, W., Jr., N. Degrees of Separation: Multi-Dimensional Separation of Concerns. ICSE, 1999.

To Enlighten Hidden Facts in The Code: A Review of Software Visualization Metaphors

Yangyang XU*, Yan LIU† and Jiabin ZHENG‡
School of Software Engineering, Tongji University
Shanghai, China

Email: *1334902@tongji.edu.cn, †yanliu.sse@tongji.edu.cn, ‡1434321@tongji.edu.cn

Abstract—Software visualization has been adopted to help engineers understand the design and functionality better and faster. A number of visualization techniques have been developed in the field of structure, behavior and evolution recently. However, there is little attempt to comprehensively review current state of the art for software professionals. As a consequence, this paper employs a systematic review of research literature on the visualization of code, to identify current application tasks, discuss variety effectiveness of visual representations, and sort out their relationships to improve usability. Finally, unsolved issues and future research opportunities have been discussed.

Keywords—code visualization; metaphor; mapping; systematic literature review;

I. INTRODUCTION

Software systems have increasingly grown in size and complexity. Due to the high turnover rate and changing industry environments, engineers have encountered challenge in software comprehension and maintenance. Particularly, studies indicate that 80% of the software costs are used for maintenance, in which 40% is devoted to understand source code[1]. It is recognized that people are better at deducing information from graphical image than numerical and textual formats[2]. Therefore, Software visualization(SV) is defined as a technique which transfers hidden facts into visual forms like images, diagrams or animations[3]. However, most of the current research focuses on specific technique of analysis, there is little work on how visualization techniques really facilitate general tasks for stakeholders. Critically, the challenge is to find a good visual structure which maintains fulfilled information of tasks and can be perceived easily. As a consequence, SV techniques have not been widely adopted in the industry[4].

The body of work on SV is such that, at first, it is important to identify visualization tasks and select adaptive metaphors. This is because SV should always be goal-oriented[5]. Namely, visualization goals drive the definition of SV techniques. Efforts on this research can be categorized in two approaches: empirical study and literature review. With the limitation of quantitative analysis for SV, we decided to conduct a systematic literature review(SLR) of software code visualization. On the basis of running through state of the art, we could summarize existing visual metaphors, understand analytic tasks and obtain a better mapping between them. The results are expected to be utilized as a foundation for potential experimentation and professional scholars.

TABLE I. Research questions of SLR

Number	Research question
RQ1	What analytic tasks does current SV support?
RQ2	What types of perspectives do engineers use SV techniques?
RQ3	What types of visual metaphors are available in the study?
RQ4	Which tools are used to support software code visualization?
RQ5	What is the correlation between tasks and visual metaphors?

II. RESEARCH METHODS

A. Planning the Review

Prior to undertaking a SLR it is necessary to identify the purpose[6], namely, to explore relevant literature through research question “how visualization metaphors facilitate tasks supported in current techniques?”

1) *Research questions*: Relevant research questions(RQs) to guide SLR have been formulated in Table I.

The motivation of RQ1 and RQ2 is to identify the goal of SV techniques. RQ3 and RQ4 get a comprehensive set of available visualization techniques. The objective of RQ5 is to investigate relationship between techniques and supported tasks.

2) *Review protocols*: With the definition of RQs, it is essential to specify review protocols to reduce possibility of bias[6]. It includes search terms, search resources, selection criteria and data extraction strategy.

In this SLR, initial search terms were “software visualization” and “visualization techniques”. Moreover, “visual”, “visualize” and other synonyms could be considered as search terms. Query of this review were built mainly in 6 top publication venues of SV area, along with research in 4 representative databases: IEEE Xplore, Scencedirect, ACM Digital Library and Springer Link.

Selection criteria are listed in Table II. Data items to extract related information are defined as follows: analytic

TABLE II. Inclusion and exclusion criteria of SLR

Inclusion criteria	
1	A study is published after 2007.
2	A study discusses about SV supported tasks, metaphors, tools, and evaluation.
Exclusion criteria	
1	A study does not include code visualization.
2	A study is with little evidence or outdated.
3	A study is duplicate.

TABLE III. Number of paper per step for per venue

Public venue	Search	Selection1	Selection2
ICSE	115	89	12
ICPC	79	70	10
ICSM	77	48	5
SoftVis	96	14	2
WCRE	68	50	7
WICSA	9	8	1

tasks/visualization activities for RQ1, RQ2; visualization representation/metaphors for RQ3; tool support for RQ4; and conclusion/relationship/correlation for RQ5.

B. Conduct the Review

SV papers have been published in many venues. We selected 6 representative ones as paper sources, including ICSE, ICPC, ICSM, SoftVis, WCRE and WICSA. Table III indicates selected number of papers in each step for each venue. Initial results were achieved with query of search terms. Due to the large number of papers, we applied selection criteria in Table II to concentrate the results. We limited the date of publication to consult mature theory in the first selection. And manual search method was performed in the second round. As a result, 37 papers were selected from these venues. In addition, same research method has been explored in 4 famous public databases and we retrieved 50 results. After removing duplicate ones within two approaches, finally we identified 81 papers for the review. Due to the limited space, selected papers are listed in “http://SSE.tongji.edu.cn/liuyan/sv_papers.html”.

III. VISUALIZATION REVIEW RESULT

After conducting the review, this section reports results based on the synthesis and analysis of data extraction activities to answer RQs.

A. Supported Tasks

Software visualization focuses on diverse aspects through development stages[7]. Selected papers have indicated an increasing interest in not only visualization of software components, their properties, relationships, but also their evolution, behavior and instruction execution[1][3][7]. In order to present various tasks clearly, this review adopted the classification proposed by Stephan Diehl[3], concerning with visualizing static analysis, dynamic execution of program and evolution of code.

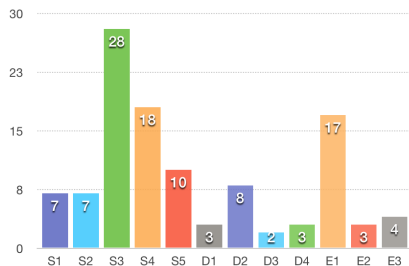


Figure 1. Number of studies per category

1) *Static analysis*: According to statistics, 54 studies in our selection have reported this aspect, including control-flow analysis[3], code map[8], dependency relationship between software components[9], software architecture[9] and code metrics[7].

Control-flow analysis (S1): It is used to depict sequential order of the program in source code which helps developers to think in an orderly manner[10].

Code map (S2): This visualization maps the relationships between pieces of code. Recent studies have indicated full interests in this analytic task.

Dependency relationship (S3): This is an essential part of software visualization owing to tremendous amount of interactions between components. It provides a visual approach for engineers to obtain an overview of dependencies for entire solution without viewing all the files and lines of source code.

Software architecture (S4): Because the focal point is code visualization, here software architecture focuses on hierarchy. It is one of core topic in SV that is used to help engineers organize artifacts into logical, abstract groups and make sure code remains consistent with the design.

Code metrics (S5): Typical static metrics of source code include size of code, number of components and complexity. However, visualization techniques discussed in 10 studies have no great changes in the past few years.

2) *Dynamic analysis*: In contrast with static properties of source code, merely 15 studies involved in research support dynamic analysis visualization. The motivation is to present what happens at run time, concerning executed time, statement coverage, dynamic architecture and program slice.

Executed time (D1) and statement coverage (D2) are aimed to optimize system performance[3][11]. 8 studies (R9,R12,R15,R19,R25,R52,R53,R64) mentioned visualization of code coverage which helps users pay more attention on frequent lines and non-executed ones.

Dynamic architecture (D3): Behavior diagrams are generated to describe changes at the level of architecture.

Program slice (D4): A dynamic slice is the set of all program points that actually affect a program point for a given input(R7,R11,R13). This is intended to find patterns in source code, and users can eliminate and sort procedures based on whether or not they are in the slice.

3) *Evolution*: Tracking changes between different versions can be meaningful for code management and maintenance[12].

Evolution metrics (E1) in this aspect include who edited specific parts of code; when each line was last modified; where bugs were located, who fixed these bugs and how the evolution processed. As a consequence, the visualization can be used in the field of code discovery and code decay[13]. Visualization of software archives (E2) which comes up from a whole overview of system updating has been reported in 3 papers. Added lines, deleted lines and changed lines reflect which class is added or removed in one file version. At the same time, visualization techniques to depict structural changes (E3) are limited[7][13].

Above all, Fig. 1 reveals that majority of selected studies investigate visualization of static aspects especially in dependency relationships and software architecture. While dynamic analysis visualization is a broad and relatively young research field due to the limitations of implementation techniques. Visualization of evolution has attracted great interests in recent year. Nevertheless, most of the analyzed work intends to visualize evolution metrics with fewer on complex structural change.

B. Principal Visual Representation

The focal step of the visualization process is to choose effective visual representations for facts in source code. They are built from points, lines, areas, and volumes with various properties: size, length, width, height, volume, position, orientation, angle, slope, color, grayscale, texture, and shape[3].

To better address RQ3, we have tried to combine representations which use different terms with same essence. Those rarely used or supported with little evidence have been excluded. Therefore, Fig. 2 presents various types of visual representations that are currently used in code visualization, ranging from simple to complex.

1) *Line representation*: Relevant visual representations include pie and bar graph, histograms and pixel representation. It makes the entire file visible with the attributes of length, indentation and color. According to the effective technique of color-coding which is beneficial for layering information, it is possible to show a million lines of code in a screen and make it easy to find different parts. However, it might be shrunk to a single row of pixels which is less readable with large scale programs.

2) *Node-link layout*: This is the most well-known metaphor to represent the relationship and hierarchy of software components[1]. It uses nodes and links to represent elements(files, packages, classes) and structural relationships respectively. Studies in selected research have indicated that representation becomes too large due to the high interconnectivity between components[7]. Related terms such as Sunburst tree layout and hyperbolic tree layout[14] have moved to more sophisticated ones to deal with the problem.

3) *Matrix views*: It is an effective visualization to display two-dimensional grid with rows corresponding to one index and columns to another[15]. In contrast to graph-based visualization, this representation provides complementary information for large programs with no overplotting. This is owing to

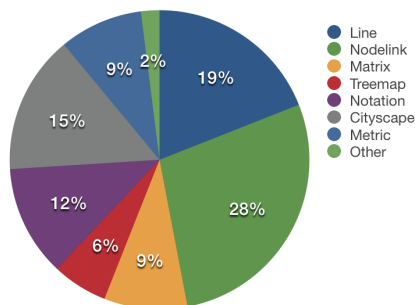


Figure 2. Number of studies per metaphor

strength of matrix that a single image contains thousands of cells[15]. One weakness of the representation is that adjacent modules are unordered and the display is irrelevant to the structure of source code tree.

4) *Treemap*: The metaphor is an effective means to visualize hierarchical decompositions of software. It executes tiling algorithm to slice the view into several parts corresponding to the number of subsystem. The space-filling technique visualizes methods as elementary boxes and classes as composed boxes which helps to address space problem in comparison with node-link graph. However, it has common problem with matrix views which is impossible to map the structure.

5) *Notation views*: This type of visualization presents the relationship between elements in a structure which is currently used for UML diagram. In contrast to node-link layout, it is reported that type of the nodes is important information in notation views.

6) *Cityscape views*: 15 studies reported this type of visualization which uses physical contexts to represent software components and relationships[16]. Similiar metaphors include forest metaphor (R39) and network view. Comparing with matrix views, it provides more intuitive view for users[17] and enhances the visualization of metrics. Nevertheless, weaknesses of 3D technology like object occlusion and performance issues have limited the usability.

7) *Metric views*: The implementation of metric view displays information on the top of UML diagram. UML is an visual modeling language to specify, design and construct software systems[18]. An extension metaphor of metric views is “areas of interest” proposed by Byelas and Telea[19].

Visualization representations proposed in research range from simple “line representation” to complex n-dimensional visualization and even animation. We do not describe these visual representations in detail, rather we consider factors that relate software visualizations to particular perspectives. As shown in Fig. 2, there is a significant difference in popularity of these visual representations. Node-link layout and line representation are still the most popular metaphors despite they have notable limitations in visualizing large scale program. 3D cityscape views and even n-dimensional representations have been proposed with the development of 3D technology. However, further research is needed for addressing the weakness of this metaphor.

TABLE IV. Number of papers associated to each analysis task and visual metaphor

Tasks	Line	Node link	Matrix View	Treemap	Notation	Cityscape	Metric View
control-flow analysis			1		5	1	
code map		5			3		
dependency relationship		15	5	2	3	7	2
software architecture		10	1	4		2	1
code metric	5	1			1	1	3
executed time	3						
statement coverage	6	1					
dynamic architecture		2					
program slice	1			1			
evolution metrics	5	1				3	5
software archives	2	1					
structural change		1	2			2	

C. Relationship between Analysis Task and Visual Metaphor

It makes no sense to translate mere source code information into a massive graph. The utility of visualization lies in an appropriate, understandable and effective abstraction of the data in order to present significant information[20]. An increasing number of visual metaphors have been proposed to address different concerns which have been reported above. In order to develop suitable SV techniques, it is necessary to understand the correlation between analysis task and visual metaphor.

Table IV has listed the number of studies which mentioned visual metaphors for respective tasks. There is a many-to-many relationship between analysis task and visual metaphor. As little attention has been paid on the visualization of control-flow analysis and code map, metaphors for these tasks are relatively few. Notation view is the most popular representation which has been employed to visualize control-flow of code. In contrast, visual representations have supported dependency relationships and architecture quite a lot. This is due to the importance of understanding structural elements for software maintenance. Common method to visualize relationship among components is graph. In particular, node-link layout has been widely used in dependency and architecture domains with 15 and 10 studies respectively. Because of increasing relations in code, matrix views which address space problem of node-link layout have been used for visualizing dependency in 5 studies. And cityscape views provide more vivid representation in comparison with previous two representations. Treemap is also applied to visualize relationship and software architecture, especially good at representing hierarchical information.

Few studies described the visualization of dynamic analysis. Among them, line representation is dominant metaphor which is applied for the visualization of dynamic aspect while node-link layout is rarely used.

With respect to the evolution, line representation and metric views are popular metaphors to depict change of metrics. While node-link and treemap can be used to visualize changes in structure.

IV. CONCLUSION AND FUTURE WORK

This paper presents a systematic review of code visualization which is intended to provide an understanding of current metaphors used for tasks. However, there still exists limitations for our review. First of all, we focused on English articles in six famous public venues and four large digital database which excluded several valuable literature. Secondly, the variants of search terms might cause the missing of articles. Therefore, visual concerns that we consider do not exhaust all the possibilities. Instead, they are examples to illustrate certain problems and represent most popular concepts in our review.

We have specified 3 categories of visual tasks and synthesized 7 types of metaphors from SLR. However, it is argued that what is visualized is what can be visualized, not necessarily what needs to be visualized in peer academic literature[21]. Most studies try to develop new visualization techniques as oppose as to validate or add value to existing ones. This implies importance to move from the state of the art to state of practice. Simple mapping between them has

provided theoretical evidence for deeper quantitative analysis. Consequently, to evaluate the utility of visualization metaphors and select appropriate one for specific task, next step is to make research on the higher level of abstraction with experimentation in cognitive and perceptive activities.

REFERENCES

- [1] A. Telea, L. Voinea, and H. Sassenburg, "Visual tools for software architecture understanding: A stakeholder perspective," *IEEE software*, vol. 27, no. 6, pp. 46–53, 2010.
- [2] I. Spence, "Visual psychophysics of simple graphical elements," *Journal of Experimental Psychology: Human Perception and Performance*, vol. 16, no. 4, p. 683, 1990.
- [3] S. Diehl, *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer, 2007.
- [4] H. A. Duru, M. P. Çakır, and V. İşler, "How does software visualization contribute to software comprehension? a grounded theory approach," *International Journal of Human-Computer Interaction*, vol. 29, no. 11, pp. 743–763, 2013.
- [5] V. R. Basili, J. Heidrich, M. Lindvall, J. Münch, M. Regardie, D. Rombach, C. Seaman, and A. Trendowicz, "Linking software development and business strategy through measurement," *arXiv preprint arXiv:1311.6224*, 2013.
- [6] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," Technical report, EBSE Technical Report EBSE-2007-01, Tech. Rep., 2007.
- [7] T. Khan, H. Barthel, A. Ebert, and P. Liggesmeyer, "Visualization and evolution of software architectures," in *OASiCS-OpenAccess Series in Informatics*, vol. 27. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.
- [8] "Visual studio 2013," <http://msdn.microsoft.com/zh-cn/library/dd831853.aspx>, accessed: Sept. 22, 2014.
- [9] "Visualizing and understanding code," <http://msdn.microsoft.com/zh-cn/library/dd409365.aspx>, accessed: Sept. 22, 2014.
- [10] I. Nassi and B. Shneiderman, "Flowchart techniques for structured programming," *ACM Sigplan Notices*, vol. 8, no. 8, pp. 12–26, 1973.
- [11] T. Ball and S. G. Eick, "Software visualization in the large," *Computer*, vol. 29, no. 4, pp. 33–43, 1996.
- [12] S. G. Eick, J. L. Steffen, and E. E. Sumner Jr, "Seesoft—a tool for visualizing line oriented software statistics," *Software Engineering, IEEE Transactions on*, vol. 18, no. 11, pp. 957–968, 1992.
- [13] M. Lanza, "The evolution matrix: Recovering software evolution using software visualization techniques," in *Proceedings of the 4th international workshop on principles of software evolution*. ACM, 2001, pp. 37–42.
- [14] W. Randelshofer, "Visualization of large tree structures," 2011.
- [15] S. G. Eick, T. L. Graves, A. F. Karr, A. Mockus, and P. Schuster, "Visualizing software changes," *Software Engineering, IEEE Transactions on*, vol. 28, no. 4, pp. 396–412, 2002.
- [16] R. Wetzel and M. Lanza, "Visualizing software systems as cities," in *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 2007, pp. 92–99.
- [17] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz, "Software landscapes: Visualizing the structure of large software systems," in *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*. Eurographics Association, 2004, pp. 261–266.
- [18] M. Clauß, "Generic modeling using uml extensions for variability," in *Workshop on Domain Specific Visual Languages at OOPSLA*, vol. 2001, 2001.
- [19] T. Barlow and P. Neville, "A comparison of 2-d visualizations of hierarchies," in *Information Visualization, IEEE Symposium on*. IEEE Computer Society, 2001, pp. 131–131.
- [20] M. Petre, E. de Quincey *et al.*, "A gentle overview of software visualisation," *PPIG News Letter*, pp. 1–10, 2006.
- [21] M. Petre, "Mental imagery and software visualization in high-performance software development teams," *Journal of Visual Languages & Computing*, vol. 21, no. 3, pp. 171–183, 2010.

Reliability-Based Software Rejuvenation Scheduling for Cloud-Based Systems

Jean Rahme and Haiping Xu

Computer and Information Science Department

University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA

E-mail: {jrahme, hxu}@umassd.edu

Abstract—The reliability and availability of a cloud-based system play an important role in evaluating its system performance. Due to the promised high reliability of physical facilities provided for cloud services, software faults have become a major factor for failures of cloud-based systems. In this paper, we focus on the software aging phenomenon where system performance may be progressively degraded due to exhaustion of system resources, fragmentation and accumulation of errors. We present a proactive technique, called software rejuvenation, to counteract the software aging problem. The dynamic fault tree (DFT) formalism is adopted to model the system reliability before and during a software rejuvenation process in an aging cloud-based system. Then it is converted into Markov Chains to derive the system reliability function. We use a case study of a cloud-based system to illustrate the validity of our approach. Based on the reliability analysis results, we show how to estimate software rejuvenation schedules that can keep the system reliability above a predefined critical level for required system availability.

Keywords—Software aging; software rejuvenation; reliability analysis; dynamic fault tree (DFT); Markov chain; scheduling.

I. INTRODUCTION

With the promised high reliability and availability of physical facilities, including the hardware facilities and their associated redundancy mechanisms, provided by cloud service providers, software faults have now become a major factor of cloud-based system failures. Since software reliability is considered one of the weakest points in system reliability, software fault tolerance and failure forecasting require more attentions than hardware fault tolerance in modern computer-based systems [1][2]. This work is motivated to deal with the software faults in cloud computing in order to assure high reliability and availability of cloud-based software systems. Reliability and availability are two common ways to express system fault tolerance in industry. A reliable computer-based system typically has high availability if unreliability is the major cause for unavailability. In this paper, we focus on analyzing the reliability of cloud-based systems for software fault tolerance in software reliability engineering (SRE). Traditional SRE has been based on analysis of software defects and bugs such as Bohrbugs or Heisenbugs without considering software aging related bugs [1]. The concept of software aging phenomenon was introduced in the middle 90s, which explains that the system resources used by the software degrade gradually in function of time [3][4]. Software aging

starts to show up due to multiple factors such as memory bloating, memory leaks, unterminated threads, data corruption, unreleased file-locks, storage space and fragmentation, and accumulation of round-off errors when running a software. Software aging has considerably changed the SRE field of study, and become a major factor for the reliability of fully tested and deployed software systems. To deal with software aging and to assure software fault tolerance, software rejuvenation process has been introduced as a proactive approach to counteracting software aging and maintaining a reliable software system [3]. Software rejuvenation involves actions such as stopping the running software occasionally, cleaning its internal state (e.g., garbage collection, flushing operating system kernel tables, and reinitializing internal data structures). The simplest way to perform software rejuvenation is to restart the application that causes the aging problem, or to reboot the whole system.

Due to the ever-growing cloud computing technology and its vast markets, the workload of a cloud-based system has increased dramatically. A heavy workload of cloud-based system will inevitably lead to more software aging problems. In this paper, we introduce an approach to developing rejuvenation schedules for cloud-based systems in order to maintain their high system reliability. In our approach, we adopt an analytical-based approach to compute the reliability of a cloud-based system using Dynamic Fault Tree (DFT). To maintain high system reliability and ensure a zero-downtime rejuvenation process, we introduce cloud-based spare parts as major software components. Once the DFT model is developed, it is converted into Continuous Time Markov Chains (CTMC) to calculate the system reliability. We assume a practical reliability threshold for the core software components of the system. When the threshold is reached, the software rejuvenation process is triggered, and the reliability of the cloud-based system is boosted to its initial state. Our case study shows that software rejuvenation scheduling based on the reliability analysis of a cloud-based system can significantly enhance its system reliability and availability.

Previous studies on software aging and software rejuvenation for predicting a rejuvenation schedule can be classified into two categories, namely analytical-based and measurement-based approaches [5]. In an analytical-based approach, a failure distribution is assumed for software faults related to the software aging phenomenon, and software rejuvenation is executed at a fixed interval based on the analytical results of the system reliability and availability [6].

Several analytic models have been proposed to determine the optimal time for rejuvenation. Bobbio *et al.* proposed a fine-grained software degradation model for optimal rejuvenation policies [7]. Based on the assumption that the current degradation level of the system can be identified, they presented two different strategies to determine whether and when to rejuvenate. Vaidyanathan *et al.* presented an analytical model of a software system using inspection-based software rejuvenation [8]. In their approach, they showed that inspection-based maintenance was advantageous in many cases over non-inspection based maintenance. Although the above approaches proposed various models for software rejuvenation, they are not intended to address complex system components' behaviors and interactions, such as dynamic relationships between software components including sparing relationship and functional dependency. Different from the existing analytical-based approaches, we focus on the dynamic behaviors of software components in the context of cloud-based systems. We use sparing relationships as an example to show how dynamic relationships of software components in a cloud-based system can be modeled using DFT.

On the other hand, measurement-based approach applies statistical analysis to the measured data of resources usage and degradation that may lead to the software aging problem. In a measurement-based approach, a monitoring program is used to continuously collect the system performance data, which are analyzed to estimate the system degradation level. When exhaustion reaches a critical level, the software rejuvenation process is triggered. Machida *et al.* used Mann-Kendall test to detect software aging from traces of computer system metrics [9]. They tested for existence of monotonic trends in time series, which are often considered indication of software aging. Grottke *et al.* studied the resource usage in a web server subject to an artificial workload [10]. They applied non-parametric statistical methods to detect and estimate trends in the data sets for predicting future resource usage and software aging issues. The existing measurement-based approaches are feasible ways to detect software aging problems in real-world computer-based systems; however, they typically involve processing of large amount of system data. Thus, they are not as efficient as analytical-based approaches. On the other hand, measurement-based approaches do provide useful insights about the system behaviors and failure distributions related to software aging. As such, our research is complementary to research efforts that investigate the relationships of static features of software and metrics for software faults with the software aging phenomenon using statistical analysis.

II. REJUVENATION OF CLOUD-BASED COMPONENTS

In a cloud-based system, virtualization allows one to share a machine's physical resources among multiple virtual environments, called virtual machines (VM). As shown in Fig. 1, a VM is not bounded to the hardware directly; rather it is bounded to generic drivers that are created by a virtual machine manger (VMM) or a hypervisor. Since a VM can be easily created and destroyed, it is particularly useful in a disaster recovery process of a cloud-based system. In this paper, we refer a cloud-based system as a software system that consists of multiple VMs, where each VM is considered a software component of the cloud-based system.

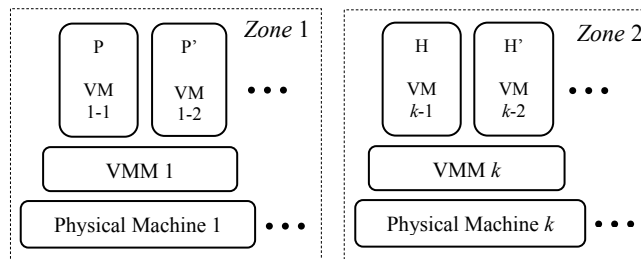


Figure 1. An example of reliable cloud-based systems

As a proactive fault management technique, software rejuvenation has been used to refresh system internal states and prevent the occurrence of software failures due to software aging. As we have mentioned, a simple way of software rejuvenation in a cloud-based system is system reboot, e.g., to restart a VM or all VMs of the system. The basic idea of our approach is to create a new instance of VM that replaces the one to be rejuvenated. Since the newly deployed VM instance has not yet been affected by the software aging phenomenon, the reliability of the software component is boosted back to its initial condition. To achieve high fault tolerance and reliability, we further adopt the software redundancy technique using two different types of software standby spares, namely Cold Spare Part (CSP) and Hot Spare Part (HSP). In the context of cloud-based systems, cold standby means that the software component is available as an image of a VM, rather than an active VM instance. Data between primary component and the spare one is regularly mirrored based on a specified schedule, e.g., multiple times a day. Since a CSP is not up running continuously and does not take any workload, its reliability approaches to 1 with a failure rate 0. Since a CSP can be started very quickly, the recovery time using CSP typically takes just a few minutes to no more than two hours. Note that a software-defined CSP is different from a hardware-based CSP in terms of its cost and efficiency. The cost of a software-defined CSP is its storage and very little CPU time; while a hardware-based CSP is a physical device that must be available all the time in order to assure fast failover [1]. Furthermore, a software-defined CSP can be started very quickly, but a hardware-based CSP typically requires manual configuration and adjustment in the event of partial or total failure.

Similarly, an HSP in the context of cloud-based systems is a hot standby VM instance. This means the software component serving as an HSP must be installed and deployed, and it must be instantly available in a case that the primary component fails. Although an HSP is deployed and running along with the primary component, it typically does not take any workload for processing user requests. To ensure fault tolerance, critical data is mirrored in near real time from the primary VM instance. This generally provides a recovery time of a few seconds in case of a failure. In our system design, each critical primary component is equipped with at least one HSP and one CSP in order to maintain the needed reliability. When calculating the system reliability, we only need to consider the primary component and its HSP; while the failure rate of a CSP is constantly 0. In the following, for simplicity, we denote a primary VM instance/component as P , which is active and processing workload, an HSP as H , which is active

but does not take any workload, and a CSP as C , which is inactive and also does not take any workload.

In our approach, a rejuvenation scheduling is based on the results of reliability modeling and analysis of a cloud-based system. When the reliability of a system component or the whole system reaches a predefined threshold, the rejuvenation process is triggered. We assume the rejuvenation process takes about 30 minutes, which is typically sufficient for starting a CSP and transfer all requests to the new VM. As a simple example illustrated in Fig. 1, suppose we have two instances, i.e., a primary component P and a hot standby one H , which are deployed on two different physical machines. The two physical machines usually belong to two different zones (denoted as Zone 1 and Zone 2 in Fig. 1), so a power/network outage in one zone, will not affect the availability of the other one. To rejuvenate the whole system, we need to start two CSPs P' and H' to replace P and H , respectively. As shown in Fig. 1, P' and H' are deployed on the same physical machine where P and H are deployed, respectively, but in reality, both P' and H' can be deployed on any physical servers.

Once the spare components P' and H' are up and running, P' will start processing new system requests, while H' is kept alive and will not take any workload. Meanwhile, we allow 30 minutes for the old components P and H to finish processing their existing requests. After 30 minutes, we shut down and delete the components P and H , which has been successfully replaced by P' and H' after completion of the rejuvenation process. Note that we do not try to restart and reuse the same instances P and H in our approach. This is because different from a physical machine, a VM can be easily created and deployed, thus deploying new instances P' and H' is a much more efficient way than restarting P and H .

During the rejuvenation procedure, we consider two scenarios. One scenario is to rejuvenate the major software components all together. In this case, we replicate the whole system at the same time when the system reliability reaches its threshold. We call this scenario as a system-specific rejuvenation. The second scenario is a component-specific one, meaning that each time, we only rejuvenate the critical component whose reliability is the lowest one when the system reliability reaches its reliability threshold. As we will show in a case study, the component-specific rejuvenation usually demonstrates certain advantages over the system-specific approach.

III. MODELING AND ANALYSIS USING DFT

In this section, we first briefly introduce DFT, then we show how to use DFT to model and analyze the reliability of a cloud-based system subject to software rejuvenation. To simplify matters, we assume that the time-to-failure for the software components (i.e., the VMs) has a probability density function that is exponentially distributed. Therefore, all VMs have constant failure rates.

A. Dynamic Fault Tree

The fault tree modeling technique was introduced in 1962 at Bell Telephone Lab, which provides a conceptual modeling approach to representing system level reliability in terms of interactions between component reliabilities [1]. Fault tree analysis (FTA) is by far the most commonly used technique

for risk and reliability analysis, where the system failure is described in terms of the failure of its components. Standard fault trees are combinatorial models and are built using static gates (e.g., AND-gate, OR-gate, and K/M-gate) and basic events. As combinatorial models can only capture the combination of events without considering the order of occurrence of their failures, they are usually inadequate to model today's complex dynamic systems.

DFT augments the standard combinatorial gates of a regular fault tree, and introduces three novel modeling capabilities, namely spare component management and allocation, functional dependency, and failure sequence dependency. These modeling capabilities are realized using three main dynamic gates: the spare gate, the functional dependency gate, and the priority-AND gate. The work done in this paper uses the dynamic spare gate, in particular the hot spare gate or HSP gate. Note that a spare gate has one primary input and one or more alternate inputs (i.e., the spares). The primary input is initially powered on, and when it fails, it is replaced by an alternate input. The spare gate fails when the primary and all the alternate inputs fail.

Since a DFT failure model is typically used to describe dynamic relationships rather than simple combinatorial ones, we need to transform it into a state-based formalism, such as Markov chains, for formal analysis. In the following section, we show how to convert a DFT model into Markov chains.

B. Modeling and Analysis Using DFT

To model and analyze the reliability of a cloud-based system with spare parts, we consider two different phases. **Phase 1** represents the pre-rejuvenation phase, where the reliability analysis is based on the failure rates of the primary components and their HSPs. CSPs are not considered for reliability analysis, as they cannot take over the system load instantly when both the primary and hot spare components fail. We model the system reliability using DFT, and then the DFT is converted into a CTMC to derive the system reliability function.

Figure 2 shows a simple hot spare gate with one primary component denoted as P and one hot spare part denoted as H . At the right-hand side of the figure, we show the CTMC model corresponding to the HSP gate. There are four states 1 to 4 defined in the CTMC model, which are denoted as PH , P , H^* , and $FAILURE$, respectively. The state PH (State 1) refers to the one in which both the primary component and the hot spare part are functioning. When the hot spare part component or the primary component fails, the model enters its P state (State 2) or H^* state (State 3), respectively.

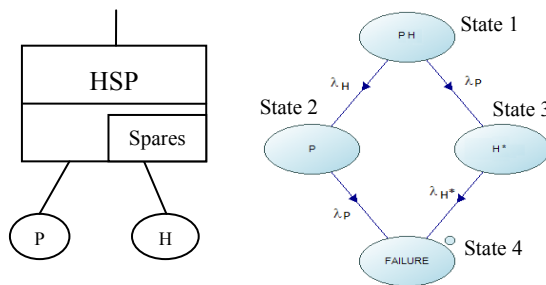


Figure 2. An HSP gate and its corresponding CTMC model

Note that we denote State 3 as H^* instead of H because in State 3, the hot spare part has a different failure rate as the one in State 1. The reason why H and H^* have different failure rates is described as follows. In State 1, the hot spare part does not take any workload, therefore its failure rate λ_H is fairly low; however, in State 3, the hot spare part takes the normal workload as the primary one before it fails, its failure rate becomes higher due to the software aging phenomenon. Suppose the hot spare part has the same configuration as the primary one, then in State 3, its failure rate shall equal to the primary component's failure rate λ_p .

Let $P_i(t)$ be the probability of the system in state i at time t , where $1 \leq i \leq 4$, and $P_{ij}(dt) = P[X(t+dt) = j | X(t) = i]$ be the incremental transition probability with random variable $X(t)$. The following matrix $[P_{ij}(dt)]$, where $1 \leq i, j \leq 4$, is the incremental one-step transition matrix [1] of the CTMC defined in Fig. 2.

$$[P_{ij}(dt)] = \begin{bmatrix} 1 - (\lambda_p + \lambda_H)dt & \lambda_H dt & \lambda_p dt & 0 \\ 0 & 1 - \lambda_p dt & 0 & \lambda_p dt \\ 0 & 0 & 1 - \lambda_{H^*} dt & \lambda_{H^*} dt \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The matrix $[P_{ij}(dt)]$, where $1 \leq i, j \leq 4$, is a stochastic matrix with each row sums to 1. This matrix provides the probabilities for each state either remaining (when $i = j$) or transit to a different state (when $i \neq j$) during the time interval dt . Given the initial probabilities of the states, the matrix can be used to describe the state transition process completely. From the matrix defined in Eq. (1), we can derive the following relations as in Eqs. (2-4).

$$P_1(t + dt) = (1 - (\lambda_p + \lambda_H)dt)P_1(t) \quad (2)$$

$$P_2(t + dt) = (\lambda_H dt)P_1(t) + (1 - \lambda_p dt)P_2(t) \quad (3)$$

$$P_3(t + dt) = (\lambda_p dt)P_1(t) + (1 - \lambda_{H^*} dt)P_3(t) \quad (4)$$

where the initial probabilities is defined as the probability of the system being at State 1; thus, we have $P_1(0) = 1$, and $P_2(0) = P_3(0) = 0$. By applying dt limit to 0, we get a set of linear first-order differential equations as in Eqs. (5-7), which are state equations for states 1-3.

$$P_1'(t) = \lim_{dt \rightarrow 0} \frac{P_1(t + dt) - P_1(t)}{dt} = -(\lambda_p + \lambda_H)P_1(t) \quad (5)$$

$$P_2'(t) = \lim_{dt \rightarrow 0} \frac{P_2(t + dt) - P_2(t)}{dt} = \lambda_H P_1(t) - \lambda_p P_2(t) \quad (6)$$

$$P_3'(t) = \lim_{dt \rightarrow 0} \frac{P_3(t + dt) - P_3(t)}{dt} = \lambda_p P_1(t) - \lambda_{H^*} P_3(t) \quad (7)$$

The state equations defined in Eqs. (5-7) can be solved using Laplace transformation, which allows transforming a linear first order differential equation into a linear algebraic equation that is easy to solve.

Let the Laplace transformation of $P_i(t)$ be $F_i(s)$, where $1 \leq i \leq 3$, we can solve the original linear first order differential equations in Eqs. (5-7) as follows.

$$F_1(s) = \frac{1}{(s + \lambda_p + \lambda_H)} \Rightarrow P_1(t) = e^{-(\lambda_p + \lambda_H)t}$$

$$F_2(s) = \frac{\lambda_H}{(s + \lambda_p + \lambda_H)(s + \lambda_p)} \Rightarrow P_2(t) = e^{-\lambda_p t} - e^{-(\lambda_p + \lambda_H)t}$$

$$F_3(s) = \frac{\lambda_p}{(s + \lambda_p + \lambda_H)(s + \lambda_{H^*})} \Rightarrow P_3(t) = \frac{\lambda_p}{\lambda_H} (e^{-(\lambda_{H^*})t} - e^{-(\lambda_p + \lambda_H)t})$$

The reliability function $R(t)$ is the summation of $P_1(t)$, $P_2(t)$ and $P_3(t)$, which can be calculated as in Eq. (8), assuming $\lambda_{H^*} = \lambda_p$, i.e., H has the same configuration as the primary one.

$$R(t) = P_1(t) + P_2(t) + P_3(t) = (1 + \frac{\lambda_p}{\lambda_H})e^{-\lambda_p t} - (\frac{\lambda_p}{\lambda_H})e^{-(\lambda_p + \lambda_H)t} \quad (8)$$

Note that $P_4(t)$ is the probability of system's being in the *FAILURE* state at time t . Therefore, the system unreliability function $U(t) = P_4(t) = 1 - R(t)$.

Phase 2 is the software rejuvenation phase. When the predefined reliability threshold is reached, the software rejuvenation process is initiated, and the system enters the software rejuvenation phase. As we have mentioned, there are two rejuvenation scenarios, namely the system-specific rejuvenation and the component-specific rejuvenation. To illustrate the basic idea of calculating reliability in this phase, we use the first scenario. In this scenario, we start two CSPs P' and H' to replace P and H , respectively. During the rejuvenation period, the four software components P , H , P' and H' coexist. As shown in Fig. 3, we decompose the dynamic fault tree model into two sub-trees, $S1$ and $S2$, which are connected by an AND-gate. Subtree $S1$ consists of the components P and H that are to be rejuvenated, while subtree $S2$ consists of the newly deployed components P' and H' , which are used to replace P and H . Both of the subtrees are defined as HSP gates, each of which can be computed using the same analysis technique as described in Phase 1. Since both of the two HSP gates are functioning at the same time, any of them fails during the rejuvenation phase will not lead to the failure of the whole system, and the system fails only when both of the two HSP gates fail. Therefore, the two HSP gates shall be connected by an AND-gate.

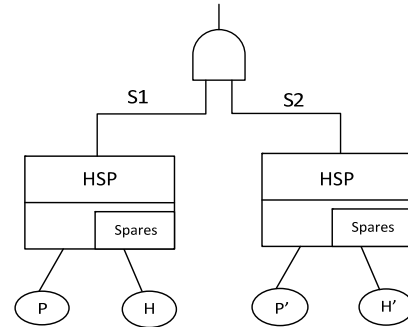


Figure 3. A DFT model with 2 HSP gates (Phase 2)

Once we have solutions to $S1$ and $S2$, the static component, i.e., the AND-gate can be easily solved using the sum-of-disjoint-products (SDP) method [1]. Specifically, to calculate the reliability of the whole system in this phase, we first calculate the unreliability functions $U_{S1}(t)$ and $U_{S2}(t)$ for $S1$ and $S2$, respectively. Then we calculate the reliability of the AND-gate as in Eq. (9).

$$R(t) = 1 - U_{AND}(t) = 1 - U_{S1}(t) * U_{S2}(t) \quad (9)$$

In the following section, we describe a case study considering both of the two scenarios during the rejuvenation process. Scenario 1 involves rejuvenation of the whole system by replicating all major software components when system reliability reaches the threshold; while in Scenario 2, we rejuvenate the most critical component with the lowest reliability when the system reliability reaches its threshold.

IV. CASE STUDY

A typical cloud-based system is illustrated in Fig. 4, which consists of an application server PA and a database server PB , all deployed on VMs. To enhance the system reliability, two hot spare components HA and HB are set up for PA and PB , respectively, which are ready to take over the workload once the primary ones fail. Note that all servers are deployed on VMs in different zones for fault-tolerance purpose. As a clarification for the reliability analysis in this case study, we view a VM with its OS, the server and the deployed services as a single software component. In addition, we only consider the reliability of the servers within the box with dashed lines, and assume the proxy server's reliability is ideal. Furthermore, we assume that the proxy server and the application server can monitor and detect failures of the application server and the database server, respectively.

To ensure a high reliability of the system, we set a reliability threshold of 0.99. For this case study, we assume the typical failure rates for the servers, where $\lambda_{PA} = 0.004$, $\lambda_{HA} = 0.0025$, $\lambda_{PB} = 0.005$, $\lambda_{HB} = 0.003$. Note that the failure rates of the hot spare parts are lower than their corresponding primary ones because the spare parts do not take any workload when the primary ones are functioning. However, when the primary servers fail, the failure rates of the hot spare parts will be increased, i.e., $\lambda_{HA}^* = \lambda_{PA} = 0.004$, $\lambda_{HB}^* = \lambda_{PB} = 0.005$. This case study involves 8 software components split into two groups. The first group consists of the four servers shown in Fig. 4. The second group consists of four CSP components that are used to replace the servers in the first group during the rejuvenation process. We name the servers in the second group as PA' , HA' , PB' , and HB' . As the CSP components are undeployed VM images, their failure rates are 0. Once deployed, they will have the same failure rates as their corresponding software components.

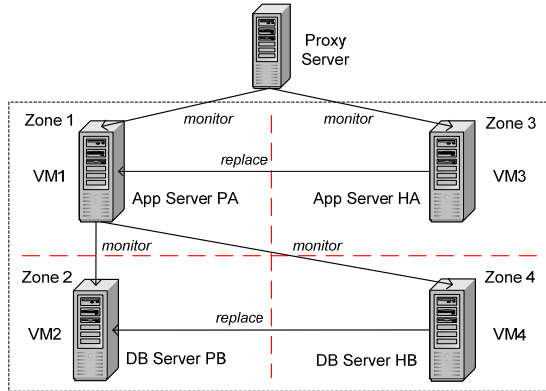


Figure 4. A cloud-based system with servers deployed on VMs

Figure 5 shows the DFT model of the cloud-based system in Phase 1. As the system fails when either the application servers fail or the database servers fail, the two HSP gates are connected by an OR-gate, which can be solved as in Eq. (10).

$$R(t) = 1 - U_{OR}(t) = 1 - (U_{S1}(t) + (1 - U_{S1}(t)) * U_{S2}(t)) \quad (10)$$

where $U_{OR}(t)$, $U_{S1}(t)$ and $U_{S2}(t)$ are the unreliability functions of the OR-gate, the subtrees $S1$ and $S2$, respectively. According to Eq. (8), $U_{S1}(t)$ and $U_{S2}(t)$ can be calculated as in Eq. (11) and Eq. (12), respectively.

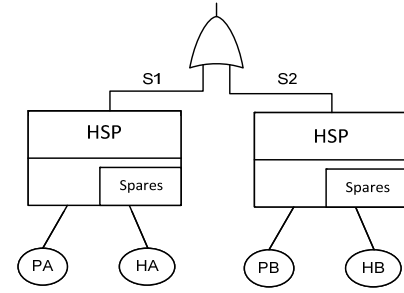


Figure 5. DFT model of the cloud-based system (Phase 1)

$$U_{S1}(t) = 1 - R_{S1}(t) = 1 - (1 + \frac{\lambda_{PA}}{\lambda_{HA}})e^{-\lambda_{PA}t} + (\frac{\lambda_{PA}}{\lambda_{HA}})e^{-(\lambda_{PA} + \lambda_{HA})t} \quad (11)$$

$$U_{S2}(t) = 1 - R_{S2}(t) = 1 - (1 + \frac{\lambda_{PB}}{\lambda_{HB}})e^{-\lambda_{PB}t} + (\frac{\lambda_{PB}}{\lambda_{HB}})e^{-(\lambda_{PB} + \lambda_{HB})t} \quad (12)$$

In Phase 2, we consider both of the scenarios mentioned in the end of Section III.B, so their impacts on system reliability as well as their consequent rejuvenation schedules can be compared. Figure 6 shows the DFT model of the cloud-based system in Phase 2 based on Scenario 1. For the same reason as in Phase 1, the system reliability can be calculated as in Eq. (13). According to Eq. (9), $U_{S3}(t)$ and $U_{S4}(t)$ can be calculated as in Eq. (14) and Eq. (15), respectively.

$$R(t) = 1 - U_{OR}(t) = 1 - (U_{S3}(t) + (1 - U_{S3}(t)) * U_{S4}(t)) \quad (13)$$

$$U_{S3}(t) = U_{S1}(t) * U_{S1'}(t) \quad (14)$$

$$U_{S4}(t) = U_{S2}(t) * U_{S2'}(t) \quad (15)$$

Note that in Eqs. (14-15), $U_{S1}(t)$, $U_{S1'}(t)$, $U_{S2}(t)$ and $U_{S2'}(t)$ can be calculated in a similar way as in Eqs. (11-12).

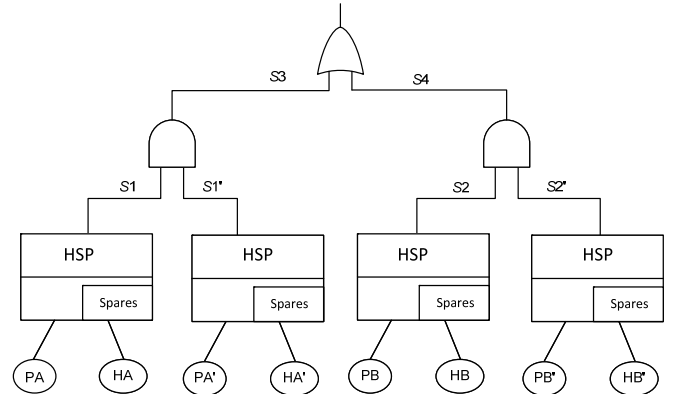


Figure 6. DFT model of the cloud-based system in Phase 2 (Scenario 1)

The reliability analysis results for Scenario 1 are listed in Table 1. The table shows that the reliability threshold (0.99) is reached every 18 days according to the reliability analysis results. Hence, both application and database servers are rejuvenated at the end of Phase 1. Phase 2 has a 30-minute time duration; therefore, we calculate the system reliability at 5, 10, 20 and 30 minutes in Phase 2 to illustrate how system reliability may change during the rejuvenation process. From the table, we can see that the system reliability is kept very high during the transition. After 30 minutes, the newly deployed servers completely take over the system, and the servers to be rejuvenated are shut down. When this happens, the system returns to its initial state, and starts a new life cycle

with a very high initial reliability. Therefore, Table 1 suggests that the system should be rejuvenated every 18 days in order to keep the system reliability above the threshold.

Table 1. System reliability with software rejuvenation (Scenario 1)

Phase	Time (Days)	App Servers Reliability	DB Servers Reliability	System Reliability
1	0	1	1	1
	1	0.99998705	0.9999801	0.9999671502577
	5	0.9996806	0.9995107	0.9991914562824
	10	0.998745	0.998085	0.9968324033250
	18	0.996044	0.994004	0.9900717201760
2	18.0035	0.99999999999	0.99999999999	0.999999999979
	18.0069	0.99999999997	0.99999999994	0.999999999917
	18.0139	0.99999999990	0.99999999977	0.999999999669
	18.0208	0.99999999978	0.99999999940	0.999999999177
1	19	0.99998705	0.9999801	0.9999671502577
	23	0.9996806	0.9995107	0.9991914562824
	28	0.998745	0.998085	0.9968324033250
	36	0.996044	0.994004	0.9900717201760
2	36.0035	0.99999999999	0.99999999999	0.999999999979
	36.0069	0.99999999997	0.99999999994	0.999999999917
	36.0139	0.99999999990	0.99999999977	0.999999999669
	36.0208	0.99999999978	0.99999999940	0.999999999177
...

By further looking into Table 1, we can see that when the system reliability reaches 0.99 after 18 days, the reliability of the database server subsystem is lower than that of the application server subsystem. This suggests that we may rejuvenate the most critical components (i.e., the component with the lowest reliability) first. Now suppose we choose to rejuvenate the database servers first. Then we wait until the system reliability reaches the threshold again, and rejuvenate the application servers next, as they now become the most critical components. This is exactly what happens for the rejuvenation scheduling in Scenario 2, where the application servers and the database servers are rejuvenated alternatively. The system reliability in Scenario 2 can be calculated in a similar way as in Scenario 1.

We now illustrate the rejuvenation scheduling for both Scenario 1 and Scenario 2 as in Fig. 7. In the figure, the start of rejuvenation is indicated by a sudden increment of the system reliability.

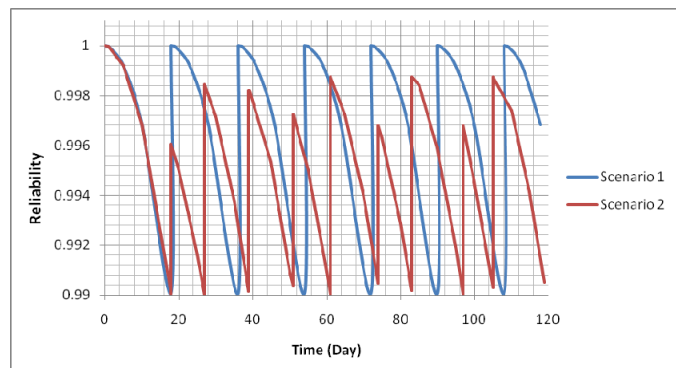


Figure 7. Rejuvenation scheduling (Scenario 1 vs. Scenario 2)

By comparing the two rejuvenation schedules, we can see that during 119 days, Scenario 1 has 6 rejuvenation process which requires rejuvenating both of the application and database servers. On the other hand, Scenario 2 has 9 rejuvenation process which only requires rejuvenating either the application servers or the database servers. It is easy to see that Scenario 2 requires less management of the servers in

order to keep the system reliability above the 0.99 threshold all the time. Suppose the rejuvenation of the application servers has the same cost as that of the database servers, by using the rejuvenation scheduling defined in Scenario 2, the cost can be reduced by $(6*2 - 9)/(6*2) = 25\%$ comparing to the rejuvenation scheduling defined in Scenario 1.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a reliability-based approach to estimating a rejuvenation scheduling for cloud-based systems. The system requires using hot spare parts during normal running time, and cold spare parts during the rejuvenation process in order to keep the system reliability above a predefined threshold. By modeling the reliability of a cloud-based system using DFT, we are able to derive the reliability function for each software component as well as the whole system. We define two phases for the software rejuvenation, and discuss about two scenarios of the rejuvenation process in Phase 2. The analysis results of our case study show that Scenario 2 is more cost-effective than Scenario 1.

For future work, we will extend our current work for components with non-constant failure rates. We will adopt a measurement-based approach to collecting empirical data in order to determine the probability density function of the system reliability affected by software aging. Software tools will also be developed for modeling and analyzing the reliability of cloud-based systems, as well as deriving effective rejuvenation schedules. Finally, modeling and analyzing cloud-based systems with active standby spare components that can share workload with the primary ones, is envisioned as a future, and more ambitious research direction.

REFERENCES

- [1] H. Pham, *System Software Reliability*, Springer Series in Reliability Engineering, Springer-Verlag London, 2006.
- [2] A. Somani and N. Vaidya, "Understanding Fault Tolerance and Reliability," *IEEE Computer*, Vol. 30, No. 4, April 1997, pp. 45-50.
- [3] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software Rejuvenation: Analysis, Module and Applications," *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, 1995, pp. 381-390.
- [4] M. Grotte, R. Matias and K. S. Trivedi, "The Fundamentals of Software Aging," *Proceedings of Workshop on Software Aging and Rejuvenation*, ISSRE, Nov. 11-14, 2008, pp. 1-6.
- [5] V. Castelli, R.E. Harper, and P. Heidelberger, *et al.*, "Proactive Management of Software Aging," *IBM Journal of Research and Development*, Vol. 45, No. 2, March 2001, pp. 311-332.
- [6] L. Jiang and G. Xu, "Modeling and Analysis of Software Aging and Software Failure," *Journal of Systems and Software*, Vol. 80, No. 4, April 2007, pp. 590-595.
- [7] A. Bobbio, M. Sereno and C. Anglano, "Fine Grained Software Degradation Models for Optimal Rejuvenation Policies," *Performance Evaluation*, Vol. 46, 2001, pp. 45-62.
- [8] K. Vaidyanathan, D. Selvamuthu and K. S. Trivedi, "Analysis of Inspection-Based Preventive Maintenance in Operational Software Systems," *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS 2002)*, Suita, Japan, 2002, pp. 286-295.
- [9] F. Machida, A. Andrzejak, R. Matias, E. Vicente, "On the Effectiveness of Mann-Kendall Test for Detection of Software Aging," *Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Pasadena, CA, November 4-7, 2013, pp. 269-274.
- [10] M. Grottko, L. Li, K. and Vaidyanathan, *et al.*, "Analysis of Software Aging in a Web Server," *IEEE Transactions on Reliability*, Vol. 55, No. 3, 2006, pp. 411-420.

Reporting an Experience on the Establishment of a Quality Model for Systems-of-Systems

Daniel Soares Santos*, Brauner R. N. Oliveira*, Adolfo Duran[†], and Elisa Yumi Nakagawa*

*ICMC, Department of Computer Systems

University of São Paulo, São Carlos, SP, Brazil.

[†]DCC, Department of Computer Science

Fraunhofer Project Center

Federal University of Bahia, Salvador, BA, Brazil.

{danielss, brauner}@usp.br, adolfo@ufba.br, elisa@icmc.usp.br

Abstract—Currently, Systems-of-Systems (SoS) have performed an important role in diverse application domains, with representative examples in airport, military, and smart-cities, including crisis/emergency management. SoS refer to complex software-intensive systems, resulted from interoperability of independent constituent systems, performing new missions that could not be performed by any constituents working separately. For these critical systems, their quality is undoubtedly essential. However, in general, there is a lack of studies that discuss how quality has been addressed in such systems. The main contribution of this paper is to present an experience of establishing a quality model (i.e., a set of quality characteristics/attributes, sub-characteristics, and metrics) for SoS, in particular, for the crisis/emergent management domain. This quality model is based on ISO/IEC 25010 and it has also proved to be an important support to evaluate a system of this domain, however their construction must be performed with caution. Experience such as presented in this work could be repeated in other domains, contributing to improve the quality of a diversity of critical, complex SoS that are currently being built.

I. INTRODUCTION

Software-intensive systems have become increasingly large and complex and even essential to the whole society. These systems are sometimes resulting from interoperability of constituent systems that work together to provide more complex missions that could not be completed by any of these systems separately [1]. This new class of software-intensive systems has been referred as System-of-Systems (SoS) and can be found in different application domains, including medicine, airport, robotics, avionics, healthcare, and automotive [2], [3]. Currently, the development of SoS still presents great challenges for the classical software engineering [4], as they presents a set of unique characteristics. Moreover, these systems must assure high level of quality considering their use in diverse critical application domains.

In another perspective, software quality has been a research topic widely researched over the last three decades

[5]. In this context, a well-accepted way to support quality control is to adopt software quality models. A quality model intends to make the software quality better understandable and manageable. A widely known model is the ISO/IEC 25010 quality model that has become an international standard for evaluating quality of modern software-intensive systems [6]. ISO/IEC 25010 is based on the fact that the software product quality can be specified and evaluated using a hierarchical structure of quality attributes/characteristics (e.g., reliability and performance), sub-characteristics (e.g., availability and adaptability), and metrics to measure these characteristics and sub-characteristics [7]. Specifically for the context of SoS, in spite of the great necessity of dealing with software quality during their development and evolution, there are not quality models for SoS that can contribute to control and improve the quality of these systems. Additionally, due to the generic nature of the ISO/IEC standard, hierarchical quality models can be tailored upon its characteristics [8].

Motivated by this scenario, the main contribution of this paper is to present an experience in establishing a quality model for SoS in the crisis/emergency management domain from the ISO/IEC 25010. This quality model has been built in the context of a large international research project — the RESCUER project¹ — which has as a main goal to develop an SoS for that domain, which intends to bring a clear impact and direct, innovative benefits to the society. Additionally, we conducted an evaluation of this SoS using this quality model. Results achieved until now show the valorous role of a quality model to improve quality of SoS. However, the building a quality model for this domain must be performed with attention mainly when it is strictly based in a standard such as the ISO/IEC 25010.

The remainder of this paper is organized as follows. Section II presents the background on SoS and quality models. Section III presents the establishment of the quality model. Section IV presents the application of this model. Section V discusses on learned lessons. Finally, Section VI presents our conclusions and future work.

DOI reference number: 10.18293/SEKE2015-155

¹<http://www.rescuer-project.org/>

II. BACKGROUND

This section presents important concepts related to SoS and quality models, aiming a better understanding about the topics covered in this paper.

Regarding to SoS, definitions and characterization of SoS have been increasingly discussed and widespread in recent years. Despite the number of different definitions existing in the literature, there is still no universally recognized definition for SoS, and then, their characterization depends often on the viewpoint and system's context. In general terms, an SoS can be seen as a "supersystem" composed by complex and operationally independent systems working together to achieve higher goal [9]. According to Maier [1], an SoS can be identified and differentiated from monolithic systems by the presence of features such as: (i) Operational Independence: constituent systems are operationally independent and have their own goals, even when disconnected from the SoS; (ii) Managerial Independence: it means that each constituent system is developed and maintained by different organizations, with their own stakeholders, development teams, processes and resources; (iii) Evolutionary Development: each constituent system evolves independently and, therefore, the SoS must also evolve, where structures, functions, and purposes are added, removed, and modified; (iv) Emergent Behavior: it means that a new behavior that can not be provided by any constituent system working separately emerges; and (v) Geographical Distribution: constituent systems may be located in different places changing information among them.

SoS started to gain popularity mainly on military systems as a strategy to reach goals or deliver unique capabilities wherein a collaborative work of complex systems is needed [1], [4]. Furthermore, SoS is migrating from traditional military domains to civil domains, such as smart homes, healthcare, crisis/emergency management, and several others. In particular, an SoS in the context of crisis/emergency management allows more efficient response to crisis and incidents through integration of police, firefighters, military, and medical systems.

Achieving quality in these systems is a quite difficult task, mainly because constituent systems are sometimes developed and maintained by different organizations, with their own stakeholders, development teams, processes, and resources [10], [11]. In this context, quality models could be used to identify relevant quality characteristics that can be further used to guide the development, evolution, and evaluation of these systems [6].

A software quality model may support a better understanding about what quality is in the context of software systems, supporting diverse activities throughout system development cycle. This is done through the identification of quality characteristics that are exhibited by software systems and can be aggregated to compose the overall software quality concept. These characteristics are generally

called quality attributes (e.g., maintainability, performance, and security), which are presented by quality models to define, assess and/or predict software quality [5]. The first models emerged in the early days of the software engineering area and since then, quality models are still subject of research. The first standardized quality model was proposed in the international standard ISO/IEC 9126-1 [12] in 2001, which was revised and replaced by the ISO/IEC 25010 [7] in 2011.

The ISO/IEC 25010 provides two quality models: (i) a quality in use model that provides five quality characteristics concerning software under operation by its stakeholders and (ii) a product quality model that is composed of eight quality characteristics concerning the software system apart of its stakeholders. Both are supposed to be applied to any kind of computer system that includes a software product. Characteristics of both models are decomposed into sub-characteristics that can be measured. When every sub-characteristic is measured, it is considered that the characteristic is also measured by aggregation. Having every characteristic measured, the overall quality of the product is determined. In order to achieve this goal, one or more metrics are defined and applied to each sub-characteristic, resulting in a value that represents the degree to which it is present in the final product. The ISO 9126-2 [13] is an example of standard that presents metrics for measuring sub-characteristics, and may be used together with ISO 25010 to evaluate quality of a software product.

III. ESTABLISHING THE QUALITY MODEL

Before we present the development of the quality model, we present the context where this model is being established and used. In the context of crisis/emergency management, the main challenge for an Emergency Command and Control Centre is to quickly obtain contextual information to answer an emergency and ensure the correct decisions. An appropriate response is essential to attenuate the occurrence of physical injuries as well as the negative outcome to the public image of the involved organizations. Decisions based on incorrect or late information have a great potential for causing more damage.

In parallel, the everyday use of mobile devices, such as tablets and smartphones, provides an enabling technology for building new software solutions for also the emergency domain. Exploring such devices as a communication tool, the RESCUER research project proposes the development of an interoperable computer-based solution to provide Command and Control Centres with real-time contextual information related to the emergency situation in industrial areas and in large-scale events. This solution relies on the collection, combination, and aggregation of crowdsourcing information.

The RESCUER solution comprises four main constituents systems:

- Mobile Crowdsourcing Solution (**MCS**) implements suitable context-sensitive mechanisms for eyewitnesses

and operational forces carrying mobile devices to provide the Command and Control Centre with information about emergency situations, taken into consideration the behavior of people under stress situations. The users provide reports of the incidents with text, photos, and videos. Besides, the RESCUER application is able to send relevant information from device sensors without user interaction.

- Data Analysis Solution (**DAS**) is composed by algorithms that process and filter the received data (e.g., image, text, and video) to extract relevant and consolidated information. This system is responsible for fusing similar data coming from different eyewitnesses in order to extract information such as the type of incident, the position and dimensions of the affected area, people density, evacuation routes, and possible approach routes for the first responders;
- Emergency Response Toolkit (**ERTK**) provides the Command and Control Centre with updated and relevant information, in an adequate format, to support decision-making during an emergency. It applies a set of solutions to manage the analyzed data coming from the DAS and presents them in a Real Time Dashboard, using adequate visualization means; and
- Communication Infrastructure (**COM**): supports the information flow between stakeholders even when traditional communication infrastructure is overloaded, by establishing Ad Hoc network communication to propagate data between users' phones and the command centre.

In the RESCUER platform, these constituents systems are part of an Integrated Solution (**IS**) that will gather crowdsourcing information and provide relevant information to the command and control centre. Since the development and evaluation of each main constituent of the RESCUER platform is not sufficient to guarantee the quality of the whole system, the IS was considered like an independent system during the development and application of the quality model.

This project defines an iterative project lifecycle, in which each subsequent iteration builds on and improves the results of the previous one. The overall strategy divides the lifecycle into three iterations steps. Basically, the iterations have been defined according to the integration of functionality (basic functions first, integration of more complex capabilities later). This facilitates the quality management, since it allows the quality evaluation of the first results and the gradual specification and maturation of the requirements of the RESCUER solution.

In this context, it can be noted that the solution emerged from the integration of described constituents systems, as well as the context development of the RESCUER solution is considered an SoS since can be perceived all the main SoS characteristics.

As early mentioned, to support RESCUER solution development, a quality model based on the ISO/IEC 25010

[7] has been established. This quality model determines quality characteristics and sub-characteristics to be considered during the three iterations of RESCUER project as well as a set of quality metrics to measure each quality characteristic. In the next sections, the establishment of the quality models is detailed concerning the quality attributes selection and quality metrics definition.

A. Quality Attributes Selection

To determine which quality characteristics and sub-characteristics are relevant to the RESCUER solution, all its non-functional requirements as well as the project goals and scope were carefully analyzed. This analysis allowed to translate each non-functional requirement into ISO/IEC 25010 quality characteristics, taking into account the Product Quality and Quality in Use models. To support this activity, it was performed a survey with requirement teams, developers, task leaders, and project coordinators in order to assure that all selected quality attributes are appropriate and relevant regarding the requirements of RESCUER solution. In this opportunity, we verified the applicability of the metrics and the viability of the application methods proposed by the ISO/IEC 9126 (it is detailed in the next section). Additionally, suggestions of other quality attributes that could be considered in the quality model and other metrics or application methods that could be used to measure the quality attributes were also gathered. After the questionnaires were answered, a meeting was performed with the stakeholders in order to discuss the results and, consequently, to obtain a consensus about the element that will compose the quality model.

The involvement of the stakeholders was very important, since the requirements about the RESCUER solution are still being detailed in the current phase of the project. Therefore, some quality attributes can still not be directly translated from the RESCUER requirements. Moreover, this strategy allows to obtain a consensus about all elements that compose the quality model, besides to assure that the main decisions about the quality model were coherent with the systems requirements and project goals.

On the other hand, it is important to highlight that not all quality characteristics and sub-characteristics are relevant for all constituents systems and as well as the IS. Depending on the use purpose of the quality model (system specification or evaluation), and the considered evaluation subject, a different subset of characteristics/sub-characteristics can be chosen accordingly to the specific goals and objectives. In addition, as RESCUER is an ongoing research project, its requirements will be probably modified during, and thus the quality model will be improved. Therefore, other quality characteristics that were not considered in the first project iteration, such as performance and security, will be added in the following project iterations. Figure 1 presents the quality characteristics and sub-characteristics selected to compose the developed quality model.

Quality Character.	Quality Sub Character.	Metric	Purpose of the metric	Method of application	Interpretation of measured value	Artifact or Data Source
Usability	Learnability	USM3	What proportion of users can operate successfully a function without a demonstration or tutorial?	Number of users that adequately operated the functions by total number of users	The closer to 1.0, the better	User test, interview or user behavior observation
		USM4	What proportion of user can operate successfully a function after a demonstration or tutorial?	Number of users that adequately operated the functions by total number of users	The closer to 1.0, the better	User test, interview or user behavior observation

Table I: Examples of Metrics

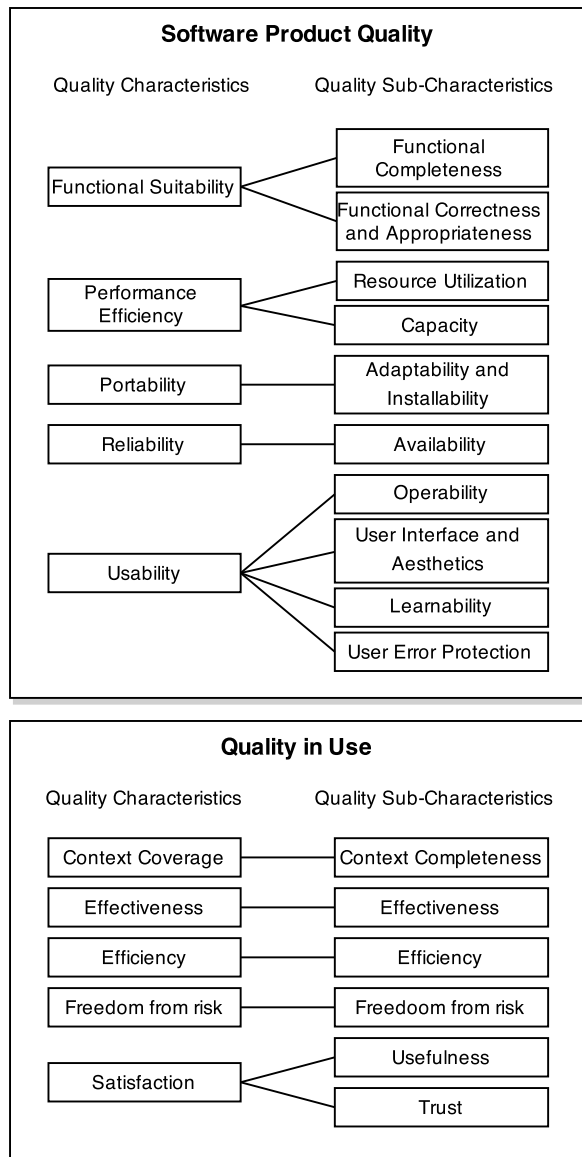


Figure 1: Established Quality Model

B. Metrics Definition

As early mentioned, quality metrics are used to measure the quality of a software product by measuring its quality attributes. When applying a quality metric, it is possible

to obtain a quantitative value that characterizes the degree of compliance of the software to the corresponding quality characteristic. In this sense, for each quality sub-characteristics defined in the quality model, a set of appropriated metrics for their measurement was established. These metrics were selected and adapted from the international standards ISO/IEC 9126-2 - External Metrics [13] and ISO/IEC 9126-4 – Quality In Use Metrics [14]. External metrics are used to measure the quality of the software product by measuring the behavior of the system, during testing stages or system operation. On the other hand, quality in use metrics are applied in a realistic system environment to verify if a product meets the needs of specified users to achieve their goals [14].

Table I presents examples of two metrics, USM3 and USM4 - USability Metric 3 and 4, of a total of 20 metrics that we have established in the quality model. These were used to measure the learnability of MCS, a key quality characteristic since no training material should be necessary for the user to understand and interact with the RESCUER system during an emergency incident, even when users are under high stress situations. In this sense, these metrics are important to identify the influence of the demonstration or tutorial in the effectiveness of the users and consequently measure the level of learnability of the RESCUER solution. In addition, it is very important to highlight that specific input data is needed for an adequate application of these metrics. Input data can be obtained by using questionnaires, checklists, experiments, observations, etc. For each established metric, the method of application and the source of data or artifacts that could be used in the measurement were established. These source of data and artifacts, as well as the strategies that will be used to obtain the needed information to application of metrics, were properly detailed in evaluation plans created to guide the constituent systems and IS evaluation. Next section will present more details about the evaluation and the application of the quality model.

IV. APPLICATION OF THE QUALITY MODEL

The developed quality model was used as basis to the evaluation of ERTK and MCS constituent systems. Each system was evaluated in four different situations, in Brazil and in Germany, taking into account the contexts of large events (FIFA World Cup 2014 and football games in Germany) and industrial areas (Camaçari Industrial

Quality Character.	Quality Sub Character.	Metric	First Iteration		Second Iteration		Third Iteration	
			Sub System	Acceptance Criteria	Sub System	Acceptance Criteria	Sub System	Acceptance Criteria
Usability	Learnability	USM3	Mobile Crowdsourcing Solution	60% of the users should adequately use the app without demonstration	Mobile Crowdsourcing Solution	65% of the users should adequately use the app without demonstration	Mobile Crowdsourcing Solution	70% of the users should adequately use the app without demonstration
		USM4	Mobile Crowdsourcing Solution	70% of the users should adequately use the app with demonstration	Mobile Crowdsourcing Solution	75% of the users should adequately use the app with demonstration	Mobile Crowdsourcing Solution	80% of the users should adequately use the app with demonstration
			Emergency Response Toolkit	N/A	Emergency Response Toolkit	50% of the users should adequately use the app with demonstration	Emergency Response Toolkit	75% of the users should adequately use the app with demonstration

Table II: Evaluation Plan

Complex² in Brazil). A total of 172 people participated of our evaluation. For this, a general evaluation plan was first developed in order to guide the evaluation of all constituents systems, including the IS, during the three iterations of the project. This general plan defines a set of assessment criteria that will be used to decide whether the metric results are satisfactory or not, considering the expected results for each project iteration. In general, these criteria are numerical thresholds or targets used to determine the need for action or further investigation. This allowed us to identify and, therefore, react in a straightforward manner to the problems that influenced the overall quality of the system. These criteria were defined through detailed analysis of the RESCUER quality requirements and refined by the requirements team, task leaders, and project coordinators.

To better manage and control the quality evolution of the RESCUER system, a different set of assessment criteria was established for each iteration. The assessment criteria were defined considering an increased level of rigor, since the metric results must improve in the course of the iterations in order to achieve the quality requirements expected to be in the final of the project. Table II presents the assessment criteria defined for the metrics USM3 and USM4, and the increasing of the rigor level of each assessment criteria during the three iterations.

For each constituent system, it was developed a specific evaluation plan to define and detail the set of strategies, source data and artifacts that will be used to obtain the needed input information for the application of the quality metrics, and, therefore, to obtain the final result about the compliance of the RESCUER constituent systems with the quality attributes established in the quality model. Since the evaluation focus of the first iteration was the usability and user experience, the strategy was basically to ask the participants to use the system following a set of key tasks while their behavior was observed in order to identify if each task was performed following an expected way. In addition, a brief user interview was performed to identify the system acceptance and aspects regarding the user experience. In the specific evaluation plan were detailed all the forms and questionnaires that were applied to evaluators and

participants in the execution of the evaluation.

In general, the application of the quality model can be summarized as following: (i) in the first step, from the quality model, it was selected the quality attributes and metrics established for the MCS addressing the first project iteration; (ii) in the second step, each quality characteristic/sub-characteristic was measured through the application of the metrics; and (iii) finally, the results were compared with the assessment criteria to identify if the quality characteristics were achieved and, consequently, to act in the quality characteristics that have not been sufficiently achieved.

Table III presents the evaluation results regarding the quality attributes defined for MCS in the first iteration of the project. The results were satisfactory considering our expectations for the first evaluation iteration of the RESCUER project. This means that, taking into consideration the average in all evaluation places, results were higher than the values of the assessment criteria. The quality model facilitated the identification of factors that can impact specific quality attributes of the system, as well as the quality of RESCUER solution as a whole. Through this first evaluation iteration and the feedback provided, it was observed that the RESCUER solution can be refined to achieve a higher quality for the next evaluation iteration and that there is still room for improvement in order to make the solution as intuitive as possible. However, the establishment of a quality model in the SoS context imposes some challenges as those described in the next section.

V. LEARNED LESSONS AND DISCUSSIONS

The establishment and use of a quality model in the SoS context impose several challenges and difficulties mainly due to the SoS characteristics, such as managerial independence, evolutionary development, and geographical distribution. In the RESCUER project, these characteristics have proven to significantly impact the productivity, success, and effort required for the establishment of a quality model. In addition, the current quality models such as the ISO/IEC 25010 have several limitations that difficult its application in the SoS context, mainly because of the lack of clearly decomposition criteria that determine how the quality attributes achieved in the constituent systems can impact and determine the SoS quality as a whole [15].

²<http://www.coficpolo.com.br/>

Quality Character.	Quality Sub Character.	Metric	Evaluation Place 1	Evaluation Place 2	Evaluation Place 3	Evaluation Place 4	Total Measure	Assessment Criteria	Total Result
Product Quality Metrics									
Usability	User Interface Aesthetics	USM2	0.84	0.87	0.81	0.87	0.84	0.70	Yes
	Learnability	USM3	0.39	0.58	0.73	0.70	0.60	0.60	Yes
		USM4	0.64	0.80	0.76	0.80	0.75	0.70	Yes
Quality in Use Metrics									
Effectiveness		ECM1	0.54	0.69	0.75	0.76	0.69	0.55	Yes
Satisfaction	Usefulness	UFM1	0.97	1.0	0.84	0.97	0.95	0.60	Yes
	Trust	TRM1	0.97	0.85	0.78	0.93	0.88	0.60	Yes

Table III: Evaluation Results of the Mobile Crowdsourcing Solution

This is a very complex problem, since quality attributes not achieved in one of their constituents can impact on the quality of other constituent systems. In addition, this impact depends on the role and importance that each constituent system plays in an SoS.

In addition, it was observed that the establishment of domain-specific quality models based on general quality models must be performed with attention. There are, for instance, domain-specific quality attributes that are not present in ISO/IEC 25010. This could be mitigated by using methodologies such as the one presented in [8], which considers external domain sub-characteristics to the general quality model.

In a parallel study, we found that the coverage rate of the ISO/IEC 25010 is only 44% regarding the quality attributes important for SoS [15]. This may significantly compromise the completeness and comprehensiveness of the quality model that have been developed. Moreover, some well-established definitions for each quality attribute can not be fully applied in the SoS context due to the flexible, dynamic nature of these systems. Therefore, some quality attributes defined in the ISO/IEC 25010 such as reliability can not directly express the required characteristics for RESCUER project and possibly others SoS.

On the other hand, it is important to say that during the refinement of the presented quality model in the next two iterations, other important key SoS quality attributes such as interoperability, security, reliability, and performance, including those ones identified in [15], will be considered.

In this sense, we expect that this quality model, with the expected improvements, can adequately guide the development and evaluation of the RESCUER solution, and our experience establishing it light the construction of quality models for domains where SoS have been applied.

VI. CONCLUSIONS

SoS is becoming increasingly important and being applied in several critical sectors of the society. By their criticality, evaluation of their quality is essential. In this scenario, this paper presented an experience of establishing a quality model for SoS, in particular, for the domain of crisis/emergency management. In addition, we applied this model in a case study to evaluate an SoS of such domain. As a result, we observed that quality models must be adopted

as one of the main guidelines to support the improvement of quality of software-intensive systems, including SoS. For the future work, we intend to apply this quality model in other evaluation iterations, as well as to update it to be consolidate as a model to be adopted for this critical, essential application domain. Besides that, we intend our experience can be reproduced in other critical domains where SoS are found.

ACKNOWLEDGMENTS

This work is supported by Brazilian funding agencies FAPESP (Grant: 2014/02244-7), CNPq (Grant: 490084/2013-3).

REFERENCES

- [1] Mark W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267 – 284, 1998.
- [2] Elisa Y. Nakagawa, Marcelo Gonçalves, Milena Guessi, Lucas B. R. Oliveira, and Flavio Oquendo. The State of the Art and Future Perspectives in Systems of Systems Software Architectures. In *SESoS*, pages 13–20, Montpellier, France, 2013.
- [3] J. A. Lane. What is a system of systems and why should i care? Technical report, USC-CSSE, 2013.
- [4] Department of Defense. Dodaf architecture framework version 2.02. <http://cio-nii.defense.gov/sites/dodaf20/>, 2010. (Accessed 20/03/2015).
- [5] S. Wagner. *Software Product Quality Control*. Springer, Berlin, Heidelberg, 2013.
- [6] N. Azizian, T. Mazzuchi, S. Sarkani, and D. F. Rico. A Framework for Evaluating Technology Readiness, System Quality, and Program Performance of U.S. DoD Acquisitions. *Syst. Eng.*, 14(4):410–426, 2011.
- [7] ISO/IEC. ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuARE) - System and software quality models. Technical report, 2011.
- [8] Xavier F. and Carvalho, J. P. Using Quality Models in software package selection. *IEEE Software*, 20(1):34–41, 2003.
- [9] M. Jamshidi. *Systems of Systems Engineering: Principles and Applications*. Taylor & Francis, 2008.
- [10] M. Gagliardi, W. Wood, J. Klein, and J. Morley. A uniform approach for system of systems architecture evaluation. *CrossTalk*, 22(3-4):12–15, 2009.
- [11] D. S. Santos, B. Oliveira, M. Guessi, F. Oquendo, M. Delamaro, and E. Y. Nakagawa. Towards the evaluation od system of systems software architecture. In *WDES*, pages 1–6, Maceio, Brasil, 2014.
- [12] ISO/IEC. ISO/IEC 9126. Software engineering – Product quality, 2001.
- [13] ISO/IEC. ISO/IEC 9126 - Software engineering - Product quality - Part 2: External metrics, 2003.
- [14] ISO/IEC. ISO/IEC 9126 - Software engineering - Product quality - Part 4: Quality in Use metrics, 2003.
- [15] D. S. Santos, T. Bianchi, K. R. Felizardo, and E. Y. Nakagawa. An investigation on quality attributes of systems-of-systems. Technical report, São Paulo, Brazil, 2015.

Experimental Frame Design Using E-DEVSML for Software Quality Evaluation

Bei Cao, Linpeng Huang, Jianpeng Hu
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China

Email: caobei.sjtu@gmail.com, Huang-lp@sjtu.edu.cn, mr@sues.edu.cn

Abstract—Quality evaluation is a critical aspect in the area of software development. If software quality problems could be found in the early design phase, the cost for software development and maintaining will be reduced. In this paper we propose an evaluation framework including a software error model and its corresponding experimental frame, which is based on Discrete Event System Specification (DEVS), to support the evaluation of multiple quality properties in the design phase. To accelerate the modeling and simulation processes, we further extend E-DEVSML to create model of system under evaluation and its experimental frame, and transform them to executable models automatically. A case study of a ticket booking system is presented to demonstrate that our approach is applicable.

Keywords—System of Systems, DEVS, quality evaluation

I. INTRODUCTION

Automated software modeling and simulation tools are by far the most promising approach to lowering the cost of software development. For successful project managements, it is very important to validate FRs and evaluate NFRs of systems precisely in early design phase before implementation of the systems [2]. Consequently the executable architectures are commonly defined to be executable dynamic simulations that are automatically or semi-automatically generated from static architecture models or products [1]. In an executable evaluation process, one of the most important issues is the design of the Experimental Frame (EF). An EF is a specification of the conditions under which a system is observed or experimented with [6]. The EF can be viewed as a system that interacts with the system of interest or system under test to obtain data under specified conditions. In this way, for an early evaluation of software quality we should concentrate on the behavior description of the system under test and the design of EF related to corresponding quality concerns, and interactions between simulated system model and EF are consequently captured for analysis and evaluation.

In this paper we propose a generic simulation approach to software quality evaluation based on a Discrete Event System Specification (DEVS) simulation framework to aid architects in the analysis of software quality attributes. First, to accelerate the modeling and simulation processes, we further extend our formal work E-DEVSML [2] to facilitate the design of experimental frame of quality evaluation. Second, an error model is given to depict behavior of the system under test for

quality concerns. At the last, we use a case of ticket booking system to demonstrate our approach.

The rest of this paper is organized as follows. Section two extends E-DEVSML for EF modeling. Section three presents our DEVS-based software quality evaluation model. A case study of a ticket booking system is present to explain how our approach worked in section four. In section five, we make our conclusion and discuss the future direction.

II. EF SPECIFICATION IN E-DEVSML

When we use DEVS to simulate a system, a critical part is to design the experimental frame (EF), which controls the simulation process. A DEVS experimental frame is often composed of three parts: acceptor, generator and transducer. The acceptor controls the beginning and end of the simulation, the generator sends requests applied to the system or model, and the transducer observes and analyzes the system output. As creating EF is a necessary part of the simulation process, reducing the complexity of writing EF programs will accelerate the DEVS modeling process. Therefore, we extend our E-DEVSML [2] to support the EF modeling. Thus EF models can be created in E-DEVSML and automatically transformed to executable languages through Xtend [4]. The detail of our method will be introduced in the following sections.

A. Acceptor

The function of an acceptor is to control the beginning and the end of simulation. The DEVS model of an acceptor is shown in Fig.1. An acceptor contains an output port to send start or stop messages. In the beginning of simulation, the acceptor stays in passive state for a certain time. After that time it sends a start message through control port and changes its state to simulating. Then the acceptor sends a stop message after a certain time simulation. The key information that is needed for us is the simulation time, wait time and the output port name which is used for the coupling with other models. The abstract syntax of an acceptor in E-DEVSML is presented in Fig. 1. We define an acceptor with keywords `acceptor`, `extends`, `waitTime`, `simTime`, and `control`. A defined acceptor can be extended by another acceptor using keyword `extends`. To make the layout of the simulation model clear, we define the acceptor as a coupled model, which contain several sub-acceptors. As its external transition function, internal transition function and output function are in the same form which can be encapsulated, we can automatically get these functions through Xtend [4].

The work described in this paper was supported by the National Natural Science Foundation of China under Grant No.61232007 and No.91118004.

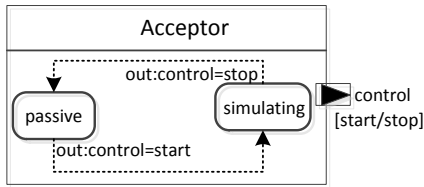


Fig. 1: Acceptor model

B. Generator

The DEVS model of a generator is shown in Fig. 2. A generator contains two ports: an input port to receive control messages for start or stop the generator, and an output port to generate requests. Requests are generated in a random way following a probability distribution. The abstract syntax of a generator is presented in Fig. 4. We define a generator with keywords generator, extends, control, out, distribution. We also defined some classical distribution types in the grammar, including poisson distribution, normal distribution and uniform distribution. We can choose an appropriate distribution suited for actual situation or define a function by ourselves.

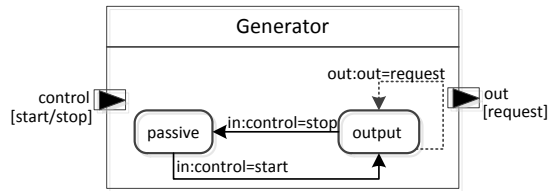


Fig. 2: Generator model

C. Transducer

A transducer receives messages generated by the simulation system and analyzes them through some calculations. The DEVS model of a transducer is shown in Fig. 3. A transducer contains several ports: an input port to receive control messages for start or stop the transducer, a set of input ports to receive messages generated by the simulation system, and a set of output ports to send the analysis results. We define the grammar of a transducer with keywords transducer, extends, vars, control, in, out in Fig. 4. Every input port of a transducer is binding with some codes which describes the behavior when a message comes to this port. Every output port of a transducer is binding to a variable which records the analysis result. The transducer is also defined as a coupled model and can be extended by another transducer.

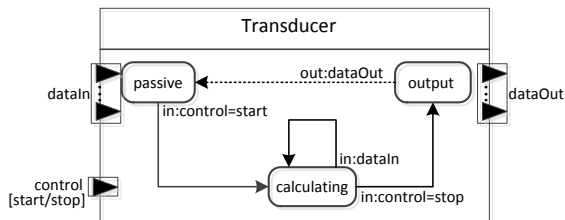


Fig. 3: Transducer model

```

E-DEVSML.xtext
Acceptor:
'acceptor' name=ID ('extends' superType=[Acceptor])?
'{' (subAC +=SubAcceptor)*'}';

SubAcceptor:
name=ID '{' 'waitTime' wt=DOUBLE 'simTime' st=DOUBLE
'control' str=STRING'}';

Generator: 'generator' name=ID
('extends' superType=Generator )?
'{' (subG +=SubGenerator)*'}';

SubGenerator: name=ID
'{' 'control' in=STRING 'out' out=STRING
'distribution' '{' distributionType=Dtype
('exp=DOUBLE', 'sigm=DOUBLE')|'double' name=ID
('') code=Code '}' '}'';

Dtype:
type=( 'Poisson' | 'Normal' | 'Uniform' );

Transducer: 'transducer' name=ID
('extends' superType=[Transducer])?
'{' (subT += SubTransducer)*'}';

SubTransducer: name=ID '{'
'vars' '{' (variables += Variable)*}'
'control' start=STRING
('in' port=STRING code=Code)*
('out' port=STRING var=[Variable])*
'}';

```

Fig. 4: The Grammar of EF Defined in EBNF

III. SOFTWARE QUALITY EVALUATION MODEL

In this section, we propose our DEVS-based software quality evaluation model. The quality evaluation model is composed of two parts: Error Model (EM) and EF. An EM describes the dynamic behavior of software or software component with parameters generated in the runtime. The EF controls the simulation process and calculates the quality metrics.

Fig.5 describes the behavior of a DEVS-based Error Model of system under evaluation with seven states: passive, active, executing, failed, recovering, error, and reboot. The passive state represents the model is waiting requests from req input port. When the request come the state changes to active. Then a message will be sent through state port. EM is in executing phase when it is processing a request. The failed state represents that a failure event happened. The recovering state represents that the model is recover from a failure. The error state represents that a fatal error event happened which leads to system reboot. The reboot state represents that the system is rebooting. An EM is in passive state before the simulations beginning. If a request comes, the state will change to active and an activated message will generate through state port to activate the failure generator and error generator. Then the state change to executing. If the request queue is not empty, the EM will keep processing the request. When a request is finished, the execution time will be sent through et port. If the request queue is empty, the state will go back to passive. If a failure comes in the executing state, the state will change to failed. After a downtime, the system starts to recover. EM comes to recover state. After a recover time, the state returns to executing. If an error comes in the executing state, the state will change to error. The system starts to reboot immediately. After a reboot time, the state returns to executing. Some messages will be produced in an internal transition as shown in Fig.5. We will capture these messages with our experimental frame.

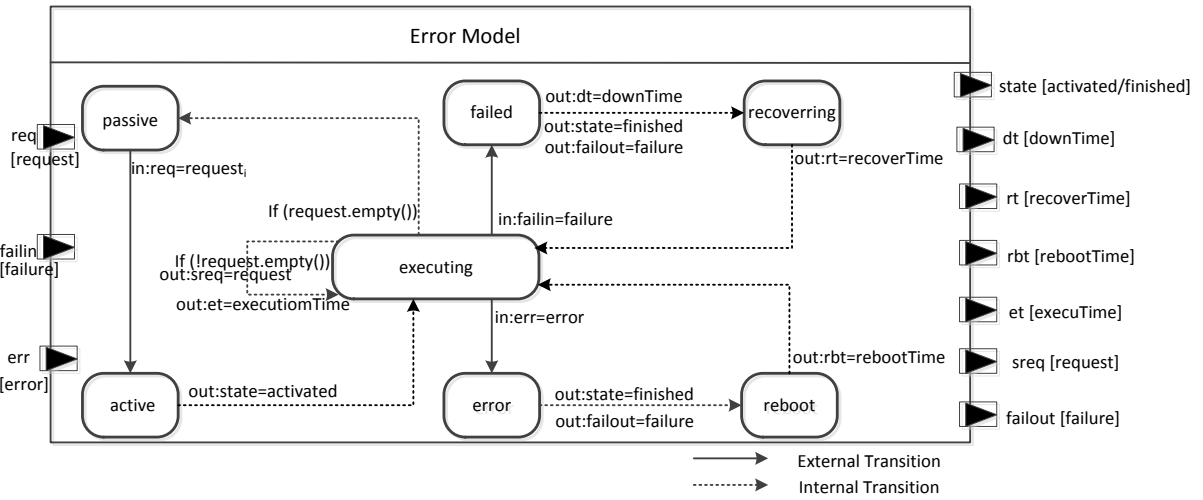


Fig. 5: Error model

We defined the corresponding EF of the error model with one acceptor, three kinds of generators and three kinds of transducers. The acceptor controls the start and the stop of the simulation. Generators are composed of request generators, failure generators, and error generators. The request generator sends request to the req port, the failure generator sends failure to failin port. The error generator sends error to the err port. Transducers are composed of performance transducers, reliability transducers and availability transducers. The performance transducer receives message from the sreq port and counts the total number of requests finished by the system. The reliability transducer receives message from failout port and counts the failures happened in the simulation process. The availability transducer receives message from port dt, rt, rbt, and et. It counts the available time and unavailable time of the system.

IV. CASE STUDY

In this section, we use a ticket booking system to explain how to design an experimental frame in E-DEVSML to evaluate software quality. Eclipse Xtext is used to specify E-DEVSML and transform E-DEVSML models to Java codes for DEVS-suite [5] which is a popular DEVS simulator. We will run the simulation and get the evaluation results in DEVS-suite.

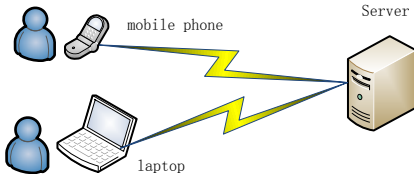


Fig. 6: A ticket booking system

A. A ticket booking system

The example ticket booking system (Fig. 6) is composed of two clients (laptop and mobile phone) and a server. Customers may use these clients to request a service, such as querying

TABLE I: Experimental settings

	mobile phone	laptop	server
etime(s)	uniform (0.3, 0.6)	uniform (0.2,0.4)	uniform (0.1,0.15)
rtime(s)	1	2	0.5
rbtime(s)	10	30	20
dtime(s)	0.3	0.2	0.1
failure rate	0.06	0.05	0.02
error rate	0.015	0.01	0.008
Request frequency	funiform (1.5, 3)	uniform (1, 2)	depends on clients

available tickets, booking movie tickets, online payment. The server answers the requests and sends the results to the clients. The server may stop providing the service if some failures or errors happen, e.g., it cannot handle any more requests or the system crashed. The clients may also experience some failures, e.g., the mobile phone lost the signal, the laptop crashed. We create DEVS models in E-DEVSML and evaluate the quality of the laptop and phone and compare the results.

B. Experiment settings

Parameters related to the quality aspects must be set before the simulation based on the behavior of previous systems, historical data and software experiences. To make the case easy to understand, we set these parameters in table I according to our experiences and set the simulation time to 3 days. In actual situation, the experimental settings could be more complex. In table I, etime represents the time to process a request, rtime represents the time needed to recover from failed state, rbtime represents the reboot time of the software system. dtime represents the down time. The word uniform represents the uniform distribution. As the clients depend on the server, the failure of server will cause the failure of clients. Although we only care about the quality properties of clients, we still have to consider the parameters of the server.

C. E-DEVSML modeling

To evaluate the quality of the system, we should create the evaluation model first. We define the ticket booking system in E-DEVSML with experimental settings. Our evaluation model

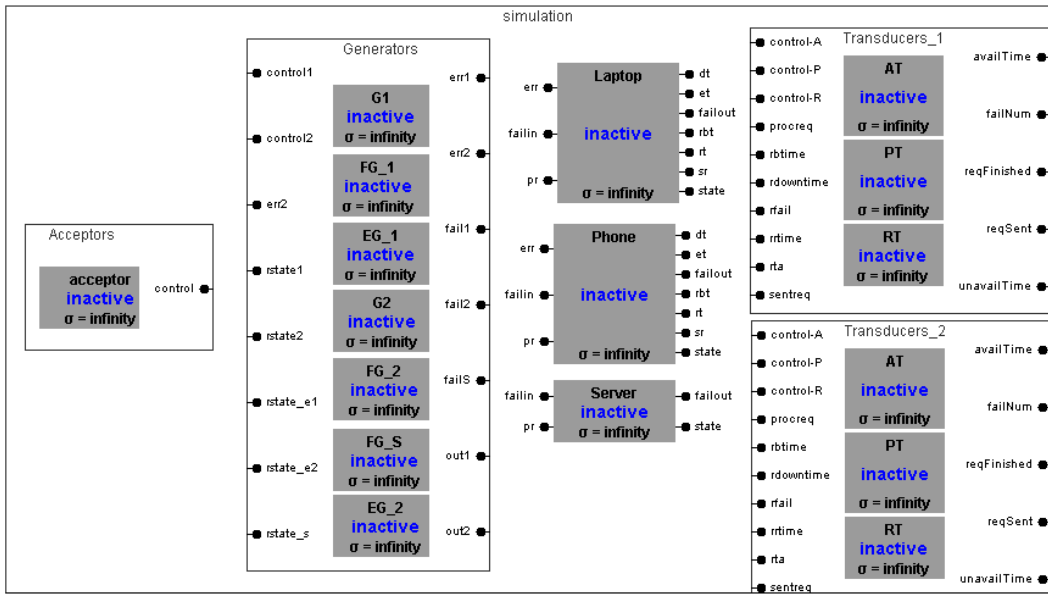


Fig. 7: Simulation model

TABLE II: Simulation results

	mobile phone	laptop
turnaround time	51319.3	50192.7
Finished requests	112155	170407
Sent requests	115274	172835
fail number	3119	2428
unavailable time	4054.7	5341.6

TABLE III: Quality properties of different clients

	Phone	Laptop
Reliability (MTBF)	83.1 s	106.8 s
Availability (available time/total time)	98.44%	99.10%
Performance (Average turnaround time)	2.31s/request	1.52s/request

is composed of three atomic models, an acceptor, a coupled generator, and two coupled transducer. We define the couplings and input ports and output ports of the simulation system in a coupled model. As E-DEVSML can be transformed to executable language through Eclipse Xtend, we define some mapping rules through Xtend and transform our E-DEVSML model to Java code which can be run on the DEVS-suite [5]. The simulation model after transformation is shown in Fig 7 .

D. Simulation results Analysis

By tracking the output ports of transducers, we can get the quality properties of the system. After the simulation, we can get the turnaround time, finished requests, sent requests, fail number and unavailable time. The results we get from these out ports are shown in table II. We can calculate the classical quality properties of the system through the data we get. For example the mean time between failures (MTBF), the availability and performance. Table III presents some quality properties calculated by transducer. By comparison of the quality properties of mobile phone and laptop system, we found that the laptop provide better service than the mobile phone.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we further extend our E-DEVSML with EF specification. To describe the behavior of the system, the error model presented here is general and representative, making the modeling and simulation of software system for quality evaluation more convenient. Our work in this paper has the following features: 1) the error model of system under evaluation considers not only the functional requirements, but also takes into account non-functional concerns, including failures and errors. 2) EF specification in E-DEVSML improves the modeling efficiency and the reusability of DEVS models. 3) The automated code generation improves the automation of model based evaluation process and reduces the burden of analyzers. Although E-DEVSML provides strong modeling abilities, the text based modeling approach also brings in complexity. Our future work is to enable transformation between some graphical modeling languages and E-DEVSML for EF modeling.

REFERENCES

- [1] Hu J, Huang L, Cao B, et al. Extended DEVSML as a Model Transformation Intermediary to Make UML Diagrams Executable[C]. SEKE, 2014.
- [2] Hu J, Huang L, Cao B, et al. Executable Modeling Approach to Service Oriented Architecture Using SoaML in Conjunction with Extended DEVSML[C]. Services Computing (SCC), 2014 IEEE International Conference on. IEEE, 2014: 243-250.
- [3] Sharma V and Trivedi K. Quantifying software performance, reliability and security: an architecture-based approach. J Syst Software 2007; 80: 493-509.
- [4] Bettini L. Implementing Domain-Specific Languages with Xtend and Xtend[M]. Packt Publishing Ltd, 2013
- [5] Kim S, Sarjoughian H S, Elamvazhuthi V. DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring[C]. Proceedings of the 2009 Spring Simulation Multiconference. Society for Computer Simulation International, 2009: 161.
- [6] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation, Academic Press, 2000.

Analysis of Risk Dependencies in Collaborative Risk Management

Catherine de L. Barchet, Luís A. L. Silva, Lisandra M. Fontoura
Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Santa Maria - UFSM
Santa Maria, Brasil
{catherine.barchet, silva.luisalvaro, lisandramf}@gmail.com

Abstract— Risk management aims to discuss the probabilities and consequences of risks on the goals of a software project. In such projects, there are dependence relationships between risks, although they are not treated yet by standard risk management practices. This paper is concerned with the analysis of risk dependencies, where these risks are assessed when multiple project stakeholders are involved in the development of collaborative risk debates. Our approach is based on a dialogue game protocol for collaborative risk management. This protocol mediates not only the discussion tasks of risk identification, risk analysis and risk planning, but also the collaborative debate regarding the identification and treatment of dependent risks. These risk management concepts are represented in a Bayesian network model for a risk management discussion situation, where alternative simulation scenarios can be proposed and tested in this probabilistic model according to discussion participants' requests. As observed in a case study, results from this process lead to the enhancement of the argumentative analysis of risk management issues developed by project stakeholders.

Keywords— Risk dependencies; Collaborative risk management; Dialogue game protocol; Bayesian networks.

I. INTRODUCTION

Software projects have a high probability of failing because their stakeholders are not involved in the development of risk management tasks [1]. Despite this fact, risk management practices are presented in the project management literature – PMBOK (Project Management Body of Knowledge) [2], and in the literature regarding the utilization and evaluation of software development processes – RUP (Rational Unified Process) [3], CMMI (Capability Maturity Model Integration) [1] and ISO/IEC 15504 [4]. These standard frameworks for risk management, which often rely on experts' judgment of a problem, define risk evaluation tasks where risks are regarded as independent events. However, when dependent risks exist in a software project, a risk may have a significant impact in another risk. The problem is that these risk dependencies are not captured in most risk management frameworks. Important information in the risk management process is being underused or even lost since the explicit analysis of such dependencies can allow project stakeholders to have means of constructing more effective strategies of risk management, reaching better decisions regarding the proposition and analysis of plans to deal with these risks, as observed in [5], for instance. The importance of dealing with risk dependencies is highlighted

We gratefully acknowledge financial support from CAPES - Brazil.

in [6] when stating that the evaluation and mitigation of risks may demand the analysis of complex networks representing risk dependency relationships. According to the CMMI standard [1] and as described in [7], the identification of cause-effect relationships between risks allows a more effective treatment of these risks since the exploitation of these relationships is likely to result in more comprehensive analysis of a risk management problem.

Past work in our research group describes a collaborative approach for risk management in software projects [8][9]. Here, we enhance this approach to deal with the collaborative identification and evaluation of risk dependencies. This is achieved through the expansion of a risk discussion protocol, which is formalized as a "dialogue game" [10] for collaborative risk management. Such kind of protocol amounts to a knowledge acquisition and representation solution for challenges that appear when there is a need of organizing the interchange of arguments that occurs when various project stakeholders collaborate on the deliberation of risk management situation. Based on this argumentation technique, the enhancement of this protocol which is discussed in this paper allows project stakeholders to debate the occurrence and impact of risk dependencies. Then, the information that is captured when these collaborative discussions are developed is utilized on the construction of a Bayesian network – BN [11] model for the assessment of a risk management problem. Through it, project stakeholders can simulate outcomes of a project using a graph representation that contain probability estimates linked to risks, risk causes and risk treatment plans. Our approach is implemented on a new version of a web-based system for collaborative risk discussion – RD System [8][9]. In summary, this work discusses an approach which aims the analysis and simulation of risk dependencies in software projects, where such dependencies are discussed collaboratively in a context where uncertainty is present.

In this paper, Section II discusses argumentation and risk management issues; Section III describes our enhanced dialogue game protocol for collaborative risk management of risk dependencies; Section IV exploits a BN model in the simulation of risk management issues; Section V discusses a case study carried out in our project and Section VI presents final remarks.

II. THE ANALYSIS OF RISK DEPENDENCIES IN COLLABORATIVE RISK MANAGEMENT

A risk consists on the effect of an uncertainty on the goals of a project [12]. In general, the goals of a risk management process are the identification, evaluation, treatment and minimization of risk items before they turn up as a threat for the successful execution of a project. As described in the CMMI standard [1], to the development of an effective risk management task it is necessary to have a process of risk identification in which project stakeholders are able to collaborate, in which there is a free and open debate of project risks. In addition, the judgment due to the group is likely to act as an aid on the assessment of risks, often promoting higher levels of trust on risk management plans constructed. Collaborative tasks of risk management can be structured by means of argumentation models [13][10], as proposed in [8][9]. According to [10], the modelling and representation of dialogues can be developed when “dialogue game” techniques are exploited in the construction of intelligent systems. A dialogue game [8] is a knowledge representation formalism which recognizes relevant moves of human interaction in debates involving two or more participants. A key task for the representation of a dialogue game is the definition of a set of locutions that can be utilized by debate participants. Each locution captures a participant intention to speak, such as to propose something, to answer a query, etc.

In the process of collaborative risk discussion, participants can identify risks that are dependent on other risks, since they are subject to the effects of other risks. For instance, the increment of the probability of a risk “A” may influence the probability of a risk “B” to occur in a project, in case the risk “B” is dependent on the risk “A”. A deeper level of dependency relationship between risks involves the identification of dependencies that are grounded on risk causes. For instance, risks “A” and “B” can have a common cause, and be dependent because of this. According to CMMI [1], the relationship between risks that are dependent on common causes can make easier the grouping of these risks, leading to risk treatment plans that deal with such common causes. In the analysis of dependent events, it is possible to highlight the exploitation of a BN approach [11], which is a model that permits to handle uncertainty along with the management of risks. This technique is founded on a qualitative analysis, where relevant information for the construction of the BN model is often obtained when project managers are involved in steps of debate. Most importantly, it is also grounded on the exploitation of quantitative probabilistic analysis which can be tested when a model for a problem situation is constructed. In practice, risk dependency relationships are represented in the BN model as arcs that link risks represented as nodes of a directed acyclic graph. In situations where risks are dependent because they have causes in common, instead of having risks that are directly

linked by dependency arcs, this dependency relationship is represented by a node representing a cause and risks that are dependent on this node. Once this information is modelled, queries can be executed in the BN graph, which is a task involving the generation of posterior probabilities utilizing probability tables modelled [14] [15].

III. COLLABORATIVE RISK MANAGEMENT DISCUSSIONS IN THE ANALYSIS OF RISK DEPENDENCIES

The Risk Discussion (RD) System is a web-based collaborative environment for the elicitation and organization of risk management debates [8][9]. This system mediates the communication between project stakeholders, and consequent standardization of risk information recorded in a project memory. In doing so, the RD System interprets and executes a dialogue game protocol for collaborative risk management [8], which controls the interaction steps between discussion participants according to a set of communication rules. This dialogue game protocol contains a set of locutions that are particularly directed to the capture and representation of typical risk management tasks, namely a) risk identification, b) risk analysis and c) risk planning. Besides of these problem-oriented locutions, the protocol is also formed by general purpose locutions, permitting the full development of multi-participant debates. Among these locutions, for instance, we can cite the “Ask”, “Inform”, “Argument pro” and “Argument con” locutions (see locutions in [8]). In this paper, we augment this protocol so that project stakeholders can develop dialogues aiming the identification, treatment and simulation of risk dependencies of a risk management problem.

The qualitative analysis of risk dependencies developed when the RD System is used involves the discussion of dependencies between risks which are proposed by debate participants. Once the collaborative identification and analysis of risk dependencies is executed, information related to such dependencies is utilized in the generation of a BN model. This model allows one to simulate probabilistic predictions of a risk management outcome in a project, using as input the pieces of evidence raised in the debate. This approach is a qualitative and quantitative form of assessing probabilities related to such risk dependencies, and obtaining risk simulation information which can be exploited by project stakeholders in the construction of stronger arguments to be submitted back in a debate. All locutions available in the protocol are relevant to develop risk dependency analysis; although the most exploited locutions are (Fig. 1 presents examples of these locutions):

Locution: `propose_risk(t , P_i)`, where t is a description of a risk, and P_i is any participant within the dialogue. It permits the statement of risk proposals in a collaborative risk management debate. **Preconditions:** There must have been utterance of the `start_discussion(t , P_i)` locution by any participant within the dialogue.



Figure 1. A fragment of a collaborative risk debate carried out in the RD System (part of a study case carried out in our project).

Locution: `propose_probability(t, Pi)`, where t is a description of a probability estimate, and Pi is any participant within the dialogue. Based on project stakeholders' experience, and augmented by discussion in which users can adjust their statements, the use of this locution permits the proposition of probability values regarding the occurrence of risks and/or risk causes. It is also utilized when users state probabilities regarding the effectiveness of risk treatment plans. Similarly, probabilities related to the occurrence of risk causes and the effectiveness of risk plans can also be collected when this locution is used. **Preconditions:** There must have been utterance of the `propose_risk(t, Pi)`, `propose_cause(t, Pi)` or `propose_plan(t, Pi)` locutions by any participant within the dialogue.

Locution: `propose_plan(t, Pi)`, where t is a description of a risk treatment plan, and Pi is any participant within the dialogue. It permits the proposition of plans to treat risks of a software project, where these plans can be directed to the treatment of risks that are dependent of other risks proposed in a debate. **Preconditions:** There must have been utterance of the `propose_risk(t, Pi)` locution by any participant within the dialogue.

Locution: `propose_dependency(t, Pi)`, where t is a statement composed of two risks previously proposed in the debate (along with the reserved word "is-dependent-on"), and Pi is any participant within the dialogue. It permits the identification of binary dependency relationships between two risks of a software project. Important, it is an indexing place for recording of other participants' arguments related to the detailed characterization and analysis of these dependencies. **Preconditions:** The `propose_risk(t, Pi)` locution should be inserted by any participant at least twice in the dialogue.

Locution: `propose_cause(t, Pi)`, where t is the description of a cause of risk, and Pi is any participant within the dialogue. It allows participants to indicate possible risk causes in a debate. **Preconditions:** There must have been utterance of the `propose_risk(t, Pi)` locution by any participant within the dialogue.

Once risk dependencies are made explicit, participants can generate a BN model for the risk management problem. Having this model, which can be imported by a standard tool for Bayesian analysis – the Netica System [16], discussion participants can evaluate the outcomes of prob-

abilistic simulations executed on this model. In a debate, these simulation activities are discussed when the following set of locutions are utilized:

Locution: generate_bayesian_network (t, Pi), where t is description automatic “The Bayesian Network model was generated”, and Pi is the RD System. It generates a graph representing the BN model in which its risk management concepts are captured through the locutions available in the dialogue protocol. This graph model is formed by proposals regarding risks, risk causes, risk plans, and probabilities linked to nodes of the BN model that represent these concepts, in addition to dependency relationships between risks. When this locution is used, the BN model for the risk management concepts available in the latest version of the debate is generated. In this case, an external representation to the RD System of the BN model is produced. **Preconditions:** There must have been utterance at least two **propose_risk**(t, Pi) locutions by any participant within the dialogue, with **proposed_plan**(t, Pi) and **propose_cause**(t, Pi) locutions associated.

Locution: propose_simulation(t, Pi), where t is a description of a simulation desired by any participant, and Pi is any participant within the dialogue. It permits the statement of probabilistic simulation scenarios as for promoting the analysis of risks along with their causes, the effectiveness of plans for reducing risks, the effects of risk dependencies, etc. To select risk management concepts to be exploited in a simulation scenario, in which probabilities for variable states can be changed according to users’ requests, debate participants can use locutions such as: “Select Plan”, “Select Cause” and “Select Risk”. To execute this simulation, users state that the probability of a certain plan to be effective should be exploited. This is described through the utilization of “Select Plan” and “Propose Probability for State” locutions, where the first locution selects a plan available in the BN model and the second locution describes that the probability of success of this plan should be changed. **Preconditions:** There must have been utterance of the **generate_bayesian_network**(t, Pi) locution by any participant within the dialogue.

Locution: select($type, t, Pi$), where $type$ can be {*risk, cause, plan*}, t is a statement of a risk, cause or plan to be select, and Pi is any participant within the dialogue. This locution permits the selection of risk management concepts represented in a BN model. Discussion participants can utilize it when making a detailed discussion of a simulation scenario. In our debate protocol, these simulations are captured and recorded as sub-trees of a “Propose simulation” node. **Preconditions:** There must have been utterance of the **propose_simulation**(t, Pi) locution by any participant within the dialogue.

Locution: propose_probability_for_state(t, Pi), where t is a description of a probability to be simulated in a state of a variable represented in a BN model, and Pi is any participant within the dialogue. It permits the proposition

of probability estimates for the different states of variables of a BN model. For instance, debate participants can state their beliefs of a risk treatment plan to have success, the belief of a risk cause to be present in a software project, and so on. Such statements are made on the grounds of the pieces of evidence users have observed in the current project or in debates of past projects. **Preconditions:** There must have been utterance of the **select**($type, t, Pi$) locution by any participant within the dialogue.

In addition to the rules that define the set of permitted protocol locutions, rules regulating the combined use of these locutions are represented in this model. These rules allow the RD System to automatically mediate the debate that occurs, regulating the conditions in which certain locutions can be utilized or not. Rule like these are also relevant on the representation of transitions between dialogue phases as, for instance, the transition from a risk identification phase to a risk dependency analysis phase, or a risk dependency simulation phase.

IV. THE EXPLOITATION OF A BAYESIAN NETWORK MODEL IN THE ANALYSIS OF RISK DEPENDENCIES

The process that leads to the construction of a BN model representing risks, risk causes, risk treatment plans, and the dependency relationships between these concepts is implemented in the RD System. It is a semi-automatic process in which the nodes and arcs of this Bayesian graph are derived from users’ arguments captured when the RD System is used. In essence, users advance these arguments along with the “Propose Risk”, “Propose Cause” and “Propose Plan” locutions. Then, this model is imported in the Netica System [16], allowing the participants of a debate to compute with probabilities the outcomes of queries executed in the BN model.

In the BN model for a problem situation, dependency relationships link risks which were proposed previously in a debate. In the RD System, these relationships can be captured when the “Propose Dependency” locution is used. Risks can also be dependent because they have common causes, as described in [1]. In a debate, these causal dependencies are not captured when users utilize the “Propose Dependency” locutions. To do so, the RD System automatically identifies pairs of risks that have common causes since these statements are captured explicitly when the “Propose Cause” is utilized by debate participants. It is relevant to notice that project stakeholders can exploit these causal dependencies when they plan mitigation actions to take advantage of these dependencies. Once plans are directed to the treatment of risk causes that are relevant to multiple risks, these plans can have a positive effect on these various risks at the same time. To capture these situations, we represent nodes for risk management plans in this BN model. As shown in Fig. 2, the risk treatment plans proposed in a collaborative risk debate are there in the BN graph, along with probability estimates regarding the effectiveness of these plans.

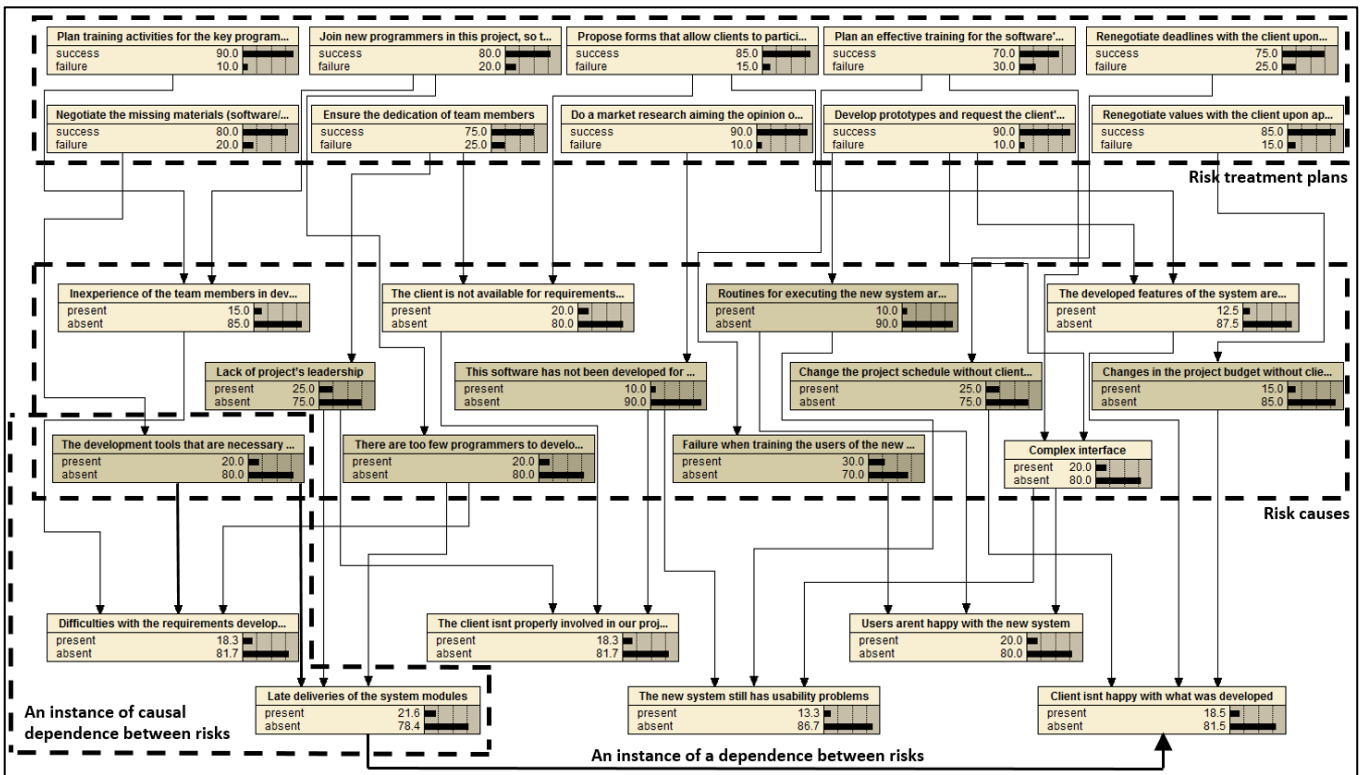


Figure 2. Topology of a Bayesian Network model generated from a collaborative risk debate (part of a study case carried out in our project).

In our BN model, risk management nodes are annotated with binary states (e.g. present and absent states of a risk variable, as in Fig. 2). Then, probability estimates are represented on the binary states of these nodes. In the Netica System, alternative queries can be executed when probability values for these variable states are input (e.g. when users input a certain probability value representing the likelihood of a risk being present in a project). This kind of input can lead to the update of the probability tables connected to the network variables of the BN model, which is something that the Netica System does automatically. In the assessment of these simulation scenarios, the RD System permits any discussion participant to utilize the “Propose Simulation”, “Select Risk”, “Select Cause”, “Select Plan” and “Propose Probability for State” locutions. Moreover, these issues can be discussed further when other general purpose locutions are utilized (see [8]) - e.g. when users discuss pros and cons of a simulation situation through the utilization of “Argument pro” and “Argument con” locutions. As a result, alternative simulation scenarios can be recorded in a risk management debate (see Fig. 1). For instance, through the combined use of the “Select Risk” and the “Propose Probability for State” locutions, users may state changes on the probability values currently linked to a selected risk. Similarly, changes on the probability values linked to risk causes and risk treatment plans can be stated in such simulation phase of a debate. Once new values of probability are presented by debate users, they can be applied in the probabilistic model which is loaded in the Netica System. This BN

model contains variables in which initial probability values are obtained from the participants’ statements collected through the use of “Propose Probability” locutions. Other probability values (e.g. posterior probabilities) are related to the outcomes produced when such probabilistic model is computed according to Bayesian rules and probability tables linked to the network variables represented in the BN model. In the simulation phase of a debate, for instance, when users suggest alternative values for probabilities of nodes representing risk treatment plans in the BN model, this action allows the risk management team to assess the impacts and consequent merits of risk management strategies. In practice, such probability suggestions aim to test different degrees of belief that users may have on the effectiveness of these plans.

V. REMARKS OF A CASE STUDY

A preliminary case study developed in our project involved the execution of collaborative tasks of risk identification, risk analysis and risk planning. In addition, a group of participants carried out tasks of identification and analysis of risk dependencies, as well as the simulation of a BN model generated from the risk management concepts proposed in a debate developed in the RD System.

The first task of this case study was the discussion of the risks of a software project. As a result, the BN model for the current risk management problem was generated – through the use of the “Generate Bayesian Network” locution. Then, the resulting probabilistic graph generated by

the RD System was imported in the Netica System. Using this model, alternative scenarios of simulation suggested were executed. In doing so, participants stated queries based on probability estimates for risks, risk causes and risk treatment plans. To do that, they utilized locutions such as “Propose Simulation”, “Select Risk” and “Propose Probability for State”, for instance. As a positive feedback from the simulation tasks executed, participants were able to formulate new arguments (e.g. new proposals for risk probability grounded on simulation outcomes) and submit them on the current debate. As an example of this simulations (see Fig. 2), it is possible to observe that the probability of the “Late deliveries of the system modules” risk is 21.6%. This estimate is quite low since the users have a plan to treat the cause of this risk (“The development tools that are necessary to develop this system are not available in this project”), and that the effectiveness of this plan is 80% as stated by debate participants. As identified collaboratively in the debate, this risk cause is also linked to the “Difficulties with the requirements development” risk, which has probability of 18.3% (very low as well). This indicates that these two risks have a causal dependency between them and the proposed treatment plan is being applied to this common cause. So, this single plan is managing to deal with two project risks at the same time, which is one of the key advantages of treating risk dependencies in risk management.

VI. CONCLUDING REMARKS

The collaborative analysis of risk dependencies in software projects is a challenging problem that is tackled superficially in standard risk management frameworks [1][2][3][4]. To approach this problem, this paper discusses the augmentation of a dialogue game protocol for collaborative risk management as presented initially in [8][9]. In general, we show how one can integrate qualitative and quantitative techniques in a collaborative risk management setting. As implemented in a new version of the RD System, this protocol now contains an expanded set of locutions along with additional risk management debate phases, which are directed to the explicit identification and analysis of risk dependencies in a software project. A key feature of this protocol is to offer knowledge acquisition and representation resources to support project stakeholders in the development of debates regarding the occurrence and effects of such risk dependencies.

The proposed approach shows how this debate protocol can guide users on the capture and recording of alternative risk dependency simulation scenarios as they are proposed and adjusted collaboratively in a debate. The debate represented when this protocol is utilized by project stakeholders can be imported in a standard BN system permitting the execution of probabilistic simulations for the investigation of dependent risks. Based on preliminary tests, the outcomes of these simulations reveal possibilities of risk management improvement allowing users to

re-estimate identified risks according to risk dependence characteristics. These improvements can be assessed not only by a small group of project managers, as commonly developed in standard risk management frameworks, but also by other project stakeholders through the proposition of new arguments in their collaborative discussions. In fact, the simulations offer feedback for the proposition of new arguments back in the debate, resulting in a qualitative enhancement of the collaborative risk management of a software project.

As future work, we plan to develop new case studies in order to obtain feedback for improving the usability of our approach. We also plan to seek connections between our collaborative risk management approach for the analysis of risk dependencies and logic-based probabilistic argumentation frameworks proposed in the literature [13], as well as making the new version of the RD System available to the public on the web.

REFERENCES

- [1] P. T. CMMI, “CMMI® for Development, Version 1.3,” *Tech. Rep. C. Carnegie Mellon Univ. Softw. Eng. Inst.*, no. November, 2010.
- [2] I. Project Management Institute, *A Guide to the Project Management Body of Knowledge*, Fifth Edit. Pennsylvania: Project Management Institute, Inc., 2013.
- [3] IBM, “Rational Unified Process (software).” IBM Rational, 2006.
- [4] ISO, “ISO/IEC 15504: Information technology - Software process assessment,” 2005.
- [5] T. W. Kwan and H. K. N. Leung, “A Risk Management Methodology for Project Risk Dependencies,” *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 635–648, Sep. 2011.
- [6] T. Alpcan and N. Bambos, “Modeling dependencies in security risk management,” *2009 Fourth Int. Conf. Risks Secur. Internet Syst. (CRISIS 2009)*, pp. 113–116, Oct. 2009.
- [7] T. O. a. Lehtinen, M. V. Mäntylä, J. Vanhanen, J. Itkonen, and C. Lassenius, “Perceived causes of software project failures – An analysis of their relationships,” *Inf. Softw. Technol.*, vol. 56, no. 6, pp. 623–643, Jun. 2014.
- [8] F. S. Severo, L. M. Fontoura, and L. A. L. Silva, “A Dialogue Game Approach to Collaborative Risk Management,” *Proc. of the 25th Int. Conf. on Software Engineering & Knowledge Engineering*, 2013.
- [9] R. C. B. Pozzebon, L. A. L. Silva, and L. Manzoni, “Argumentation Schemes for the Reuse of Argumentation Information in Collaborative Risk Management,” in *IEEE 15th International Conference on Information Reuse & Integration (IRI)*, 2014.
- [10] P. Mcburney and S. Parsons, “Dialogue Games for Agent Argumentation,” in *Argumentation in Artificial Intelligence*, G. Simari and I. Rahwan, Eds. Boston, MA: Springer US, 2009, pp. 261–280.
- [11] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. California, Los Angeles: Morgan Kaufmann, 1988.
- [12] ISO, “ISO 31000:2009 - Risk management - Principles and guidelines,” 2009.
- [13] T. Benchcapon and P. Dunne, “Argumentation in artificial intelligence,” *Artif. Intell.*, vol. 171, no. 10–15, pp. 619–641, Jul. 2007.
- [14] C.-F. Fan and Y.-C. Yu, “BBN-based software project risk management,” *J. Syst. Softw.*, vol. 73, no. 2, pp. 193–203, Oct. 2004.
- [15] Y. Hu, X. Zhang, E. W. T. Ngai, R. Cai, and M. Liu, “Software project risk analysis using Bayesian networks with causality constraints,” *Decis. Support Syst.*, vol. 56, pp. 439–449, Dec. 2013.
- [16] S. C. Norsys, “Netica Bayesian Belief Network,” 1998. [Online]. Available: <https://www.norsys.com/>.

A Practical Approach to Software Continuous Delivery Focused on Application Lifecycle Management

Everton Gomede, Rafael Thiago da Silva and Rodolfo Miranda de Barros

Department of Computer Science

State University of Londrina

Londrina, Paraná, Brazil

e-mail: {evertongomede, rafathiago}@gmail.com, rodolfo@uel.br

Abstract—To deliver quality software continuously is a challenge for many organizations. It is due to factors such as configuration management, source code control, peer-review, delivery planning, audits, compliance, continuous integration, testing, deployments, dependency management, databases migration, creation and management of testing and production environments, traceability and data post-fact, integrated management, process standardization, among others. To overcome these challenges, this paper presents a continuous delivery process that promotes artefacts produced by developers, in a managed fashion, to production environment, allowing bidirectional traceability between requirements and executables, and integrating all activities of software development using the concepts of Application Lifecycle Management. As a result, we obtained an ecosystem of tools and techniques put into production in order to support this process¹.

Keywords—Continuous Delivery; Process Quality; Application Lifecycle Management

I. INTRODUCTION

Software Delivery Process (SDP) consists of several tasks in order to promote artefacts created into the production environment (servers where an executable is installed to delivery features to the users) [1]. These ones can occur in either environment, producer or consumer. Due to the unique characteristics of each software product, a general process to various contexts probably cannot be set. Therefore, we should interpret a SDP as a *framework* to be customized according to the requirements and characteristics of each product (Software Delivery Process, in this context, is a part of Software Development Process).

This customization usually causes a *manual* execution of SDP [2]. Production environment is configured in a manual way by the infrastructure team using terminals and/or third-party tools. Artefacts are copied from a continuous integration server to a production environment and possibly some data and/or metadata are adjusted before software is released.

However, this process has some *weaknesses*. Predictability is the first one, because it increases risk, related with lost business transactions, and downtime whether any unexpected situation occurs [3]. Additionally, the repeatability factor may compromise the diagnosis of post-deployment problems [2]. Finally, this process is not auditable and it does not allow

the recovery of information about all events that were held to deliver a version.

There is a growing interest in practices to overcome these problems [4]. Such practices are known as *Software Continuous Delivery* (SCD), defined as the ability to publish software whenever necessary. This publication may be weekly, daily or every change sent to the code repository. The frequency is not important, but the *ability to deliver when it is necessary* [2].

This approach has great importance in software development because it helps who is in charge of delivering to understand better their process and, consequently, *improve it*. Such improvements can be in terms of automation, decrease of the delivery time, rework and risk reduction, or others. Among them, the main is the ability to have a version of software, ready for delivery, each new code added to the repository.

Additionally, the management of the activities of application lifecycle, support this approach and help all team members to improve their process [2], [4]. *Application Lifecycle Management* (ALM) has been proposed with the objective to provide a comprehensive technical solution for monitoring, controlling and managing software development over the whole application lifecycle [5].

Thus, we propose a question: *Are the concepts of ALM aiming at Software Continuous Delivery also an effective utility for improving the software process?* We try to answer this question by describing the results and experiences from the introduction of a Continuous Software Delivery solution complemented by techniques of Application Lifecycle Management in a financial industry company.

In this context, we present a practical approach to address the problems of software continuous delivery and application lifecycle management. The main objective is to contribute with a *setup of servers, process, techniques and tools* that assist to deliver software continuously. In addition, some recommendations and further work are discussed. Issues related with software architecture, project management and other dimensions of software development were omitted.

Therefore, this article was divided into five sections, including this introduction. In Section II, we present fundamental concepts and related works. In Section III, we present an approach to Software Continuous Delivery focused on Application Lifecycle Management. In Section IV, we present the results. Finally, in Section V, we present conclusions,

¹DOI reference number: 10.18293/SEKE2105-126

recommendations and suggestions for future work.

II. FUNDAMENTAL CONCEPTS AND RELATED WORKS

There is a relation between *quality of software products* and *quality of the process* used to build them. Implementation of a process aims to reduce rework, delivery time and increase product quality, productivity, traceability, predictability and accuracy of estimates [2]. In general, a software development process contains the activities shown in Fig. 1.

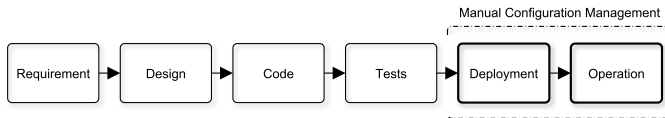


Figure 1. A simplified software development process [1], [2].

The configuration management tasks of deployment and operation activities, highlighted in Fig. 1, are usually performed manually [2]. This practice, according to Humble and Farley [2], is accompanied by anti-patterns:

- *Deploying software manually*: there should be only two tasks to perform manually; (i) choose a version and (ii) choose the environment. These are goals to be achieved in a SCD [5].
- *Deploying after development (requirement, design, code and tests) was complete*: it is necessary to integrate all activities of the development process and put stakeholders working together since the beginning of the project.
- *Manual configuration management of the production environments*: all aspects of configured environments should be applied from a version control in an automated way.

In this context, some Software Continuous Delivery Practices arises. It is a *developing discipline*, which builds up software that can be released into production at any time, by minimizing lead-time [3].

To assist this type of software delivery approach, from construction to operation, Humble and Farley presents the *Deployment Pipeline* (DP), a standard to automate the process of SCD. Despite each organization may have an implementation of this standard, in general terms, it consists of the activities shown in Fig. 2.

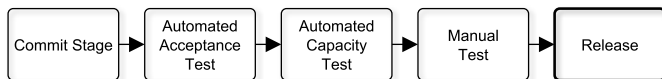


Figure 2. The deployment pipeline [2].

Over each change, artefacts are promoted to next instance of pipeline through a series of *automated tasks*. The first step of the pipeline is to create executables and installers from the code repository, in a process known as Continuous Integration (CI). Other activities perform a series of tests to ensure that the executable can be published. If the release candidate passes all tests and criteria, then it can be published [2].

To implement this pipeline, some approaches were presented. Among them, Krusche and Alperowitz [6] described the implementation of a SCD process to *multiple projects*. Their goal was to obtain the ability to publish software to their clients with just a few clicks. The main contribution of this work was to show that developers who have worked on projects with SCD, understood and applied the concepts, being convinced from the benefits of it.

Bellomo et al. [7] presented an architectural framework together with tactics to projects that address SCD. The main contribution of this work is a collection of SCD tactics in order to get software products performing with a higher level of reliability and monitoring into production environment.

Fitzgerald and Stol [4] published trends and challenges related to what the authors called “Continuous *”, which is, all topics related to software delivery that can be classified as continuous. The authors addressed issues such as; Continuous Integration (CI), Continuous Publication (PC), Continuous Testing (CT), Continuous Compliance (CC), Continuous Security (SC), Continuous Delivery (EC), among others. An important point of this paper is the distinction between the Continuous Delivery and Continuous Publication. According to the authors, Continuous Publication is ability to put into production software products in an automated manner. This definition is complementary to the software continuous delivery definition given above.

Although all these works have a practical nature, none of them showed which tools were used, which recommendations to similar scenarios and which were the techniques used during deployment. Therefore, the work presented in this paper seeks to fill these gaps.

III. A PRACTICAL APPROACH

A. Main Proposal

The Fig. 3 shows all macro elements involved in approach. In the first line, there are two ones: (i) Development, representing the timeline of software development and (ii) Operation, representing the timeline of software operation. The pipeline of SCD is between both.

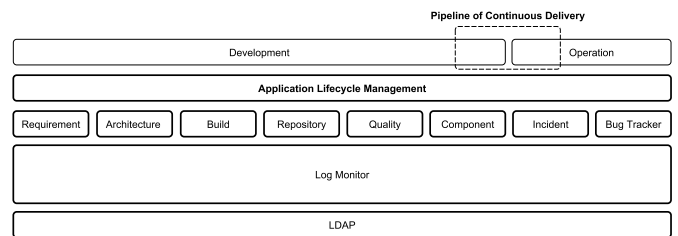


Figure 3. The big picture.

In the second and third line there are an ALM and activities of software development and operation. Requirement, architecture, build, repository, quality and component are related with development, meanwhile incident and bug tracker are related with operation. Log Monitor is used to collect and to integrate, in an automated way, information about all elements. LDAP is used to allow a *single point* of authentication and authorization between all components.

B. Infrastructure

To provide an infrastructure that allows the Software Continuous Delivery is the main goal of setup shown in Fig. 4. It has 4 areas: (i) Commit Stage (CS), (ii) Quality Assurance (QA), (iii) Staging (ST) and (iv) Production (PD).

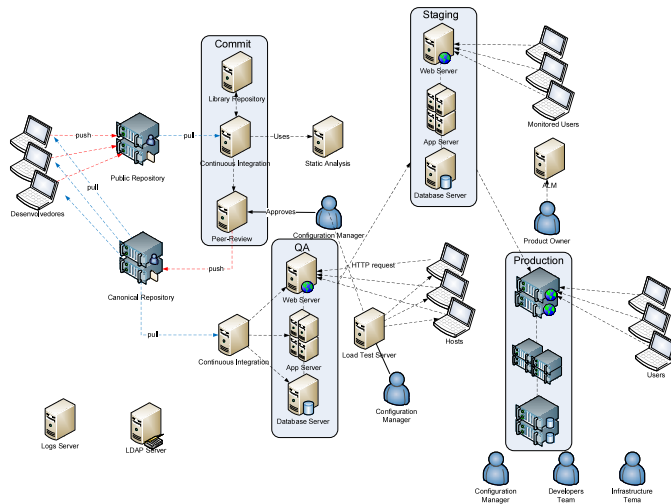


Figure 4. An overview of a setup of servers and areas.

C. Areas

The Commit Stage (CS) has primary responsibility to implement *continuous integration* of all code reviews sent to the repository. This area consists of the following services:

- **Public Code Repository**
 - Purpose: to get code reviews that have not been approved.
 - Tool: Git (git-scm.com).
 - Technique: it has a single branch, called `master`, which receives revisions of all developers.
- **Continuous Integration**
 - Purpose: to integrate all code reviews sent to the server.
 - Tool: Jenkins (jenkins-ci.org) and Maven (maven.apache.org).
 - Technique: it does integration performing unit testing and adding first acceptance step in peer-review server.
- **Static Analysis**
 - Purpose: to make code analysis generating quality reports.
 - Tool: SonarQube (sonarqube.org).
 - Technique: each integration performs a series of tests, such as size metrics, complexity, test coverage, dependency calculation, among others. Creates a baseline quality of the project.
- **Peer-Review**
 - Purpose: to enable promotion/rejection of codes from public to canonical repository.
 - Tool: Gerrit (code.google.com/p/gerrit).
 - Technique: approval of two steps, the first being carried out by continuous integration server and the second by the configuration

manager. If the review through both sides, code is promoted to canonical repository.

- **Canonical Repository**
 - Purpose: to receive approved code reviews.
 - Tool: Git (git-scm.com).
 - Technique: it has a single branch, called `master`, which receives revisions of peer-review server.
- **Repository Libraries.**
 - Purpose: to store libraries and components used in integration.
 - Tool: Nexus (sonatype.org/nexus).
 - Technique: libraries and components are installed automatically or manually on the server being available for use at the time of integration.

The layout and operation of Commit Area are shown in Fig. 5.

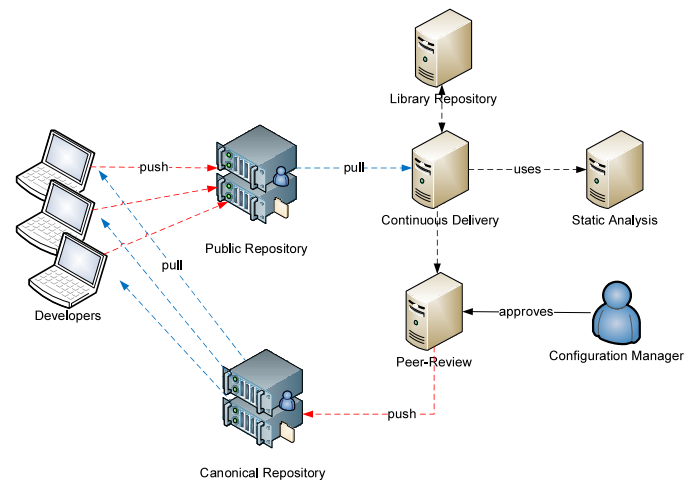


Figure 5. Commit Stage Area (CS).

The Quality Assurance Area (QA) has the main purpose of *performing all automated tests and allow Quality Manager perform manual tests*, such as exploratory testing [2]. This area consists of the following services:

- **Continuous Integration**
 - Purpose: to obtain a copy of the code and perform integration, functional and automated load tests.
 - Tool: Jenkins and Maven (maven.apache.org).
 - Technique: get a copy of canonical repository to generate executable, install them into library server, application servers and database server. After that, execute integration, functional and load tests.
- **Page Servers, Application and Database**
 - Purpose: to host application to test
 - Tools: may vary according to the technology used; Wildfly (wildfly.org) and MSSQL are some examples.
 - Technique: can vary depending on the technology used (to install and configure, basically).

- Load Test
 - Purpose: to perform a load test against the page servers, application and database.
 - Tool: Jmeter (jmeter.apache.org) and Vagrant (vagrantup.com).
 - Technique: it performs script created by quality manager allocating hosts as required to test. It generates a supported load from baseline.

The Operation of Quality Assurance area is shown in Fig. 6.

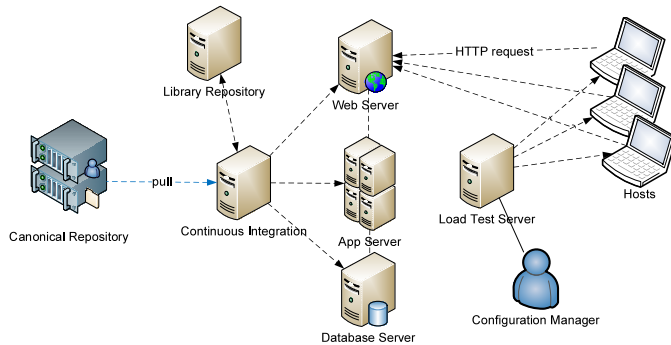


Figure 6. Quality Assurance Area (QA).

Staging Area aims to provide for monitoring users and product owners an environment as close as possible to production environment, so they perform approval tests. These ones are related to user experience and their perception regarding how software product meets specified requirements. This area has a *copy of operating environments*, both in terms of operating systems, tools and settings, and in terms of data. Monitored users are the ones chosen to perform approval tests in a monitoring way. Occasionally, they are in the product owner role. The Fig. 7 shows this area.

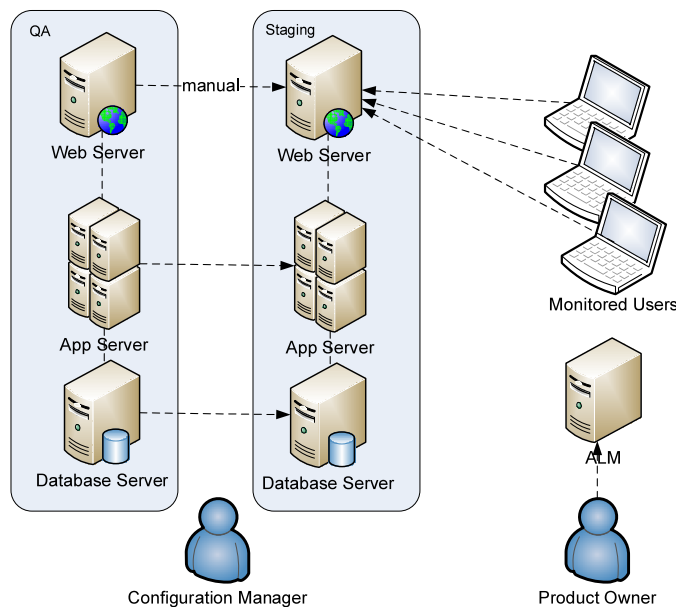


Figure 7. Staging Area (ST).

Finally, configuration manager makes promotion from Staging Area artefacts to Production Area manually by Configuration Manager. However, developers and infrastructure staff are present to perform this task. The Fig. 8 shows this area.

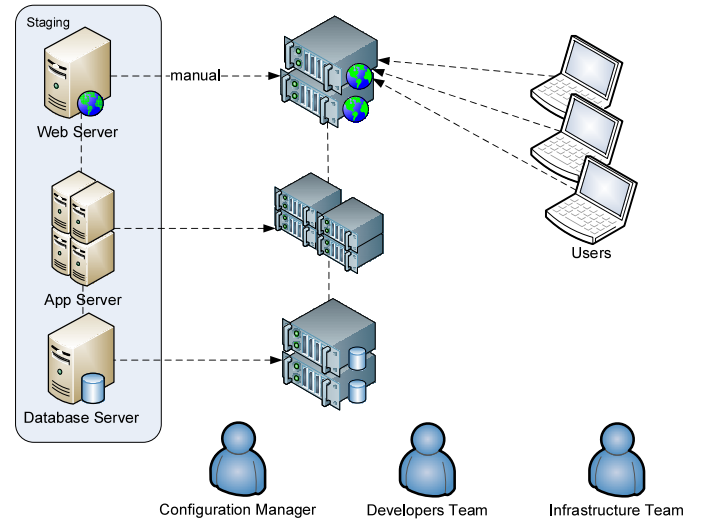


Figure 8. Production Area (PD).

Also, the following servers were used: (i) Log Server and (ii) LDAP Server. The first has a very important function in the setup; to get all events occurred by indexing logs. This assists the diagnosis, providing information to reporting, alerts and dashboard. The tool used in this case was Splunk. The second server has a function to allow authentication and authorization for all setup servers. This is necessary because it is costly to maintain users across all the servers involved in an individualized way, in addition this increase security flaws. The tool used in this case was OpenLDAP (openldap.org).

D. Tools

The Tab. I summarizes all tools used with its URL. These tools are used to Configuration Management (Git, Gerrit, Nexus, Flywaydb and Vagrant), Continuous Integration (Jenkins and Maven), Quality Assurance (SonarQube and Jmeter), Application Lifecycle Management (Redmine) and infrastructure (Splunk and OpenLDAP).

TABLE I. TOOLS USED.

Goal	Name	URL
Continuous Integration	Jenkins	jenkins-ci.org
Source Repository	Git	git-scm.com
Build	Maven	maven.apache.org
Gathering Logs	Splunk	splunk.com
Peer-Review	Gerrit	code.google.com/p/gerrit
Static Analysis	SonarQube	sonarqube.org
Load Test	Jmeter	jmeter.apache.org
Library Repository	Nexus	sonatype.org/nexus
ALM	Redmine	redmine.org
Database Migration	Flywaydb	flywaydb.org
Automated Installation	Vagrant	vagrantup.com
Authentication/Authorization	OpenLDAP	openldap.org
Architecture	Enterprise Architect	sparxsystems.com.au

These tools were used because they are open/free software or a well known tool among developers.

IV. RESULTS

Some results about this approach are related to automation of many delivery tasks, coming out in a more *predictable* and *managed* process. Another aspect, related to collaboration, is due to communication between developers and infrastructure team was increased in all aspects of the process, since planning of a feature until its publication. These results are classified in a process maturity level [2], as shown in Fig. 9.

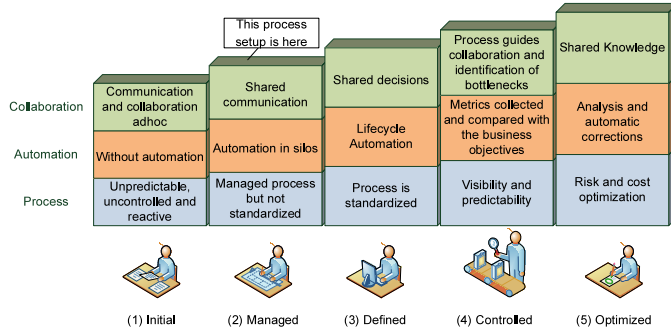


Figure 9. Process maturity level [2].

The Fig. 10 shows the peer-review authorization. It helps to protect the canonical repository and increase the quality of code for 3 reasons: (i) improve the quality of commits, each member tends to be more careful with a code that will be evaluate by another one, (ii) allow to put a sequence in commits avoiding collisions and wrong commits, (iii) allow to configuration manager to decide if the code can be promoted or not ².

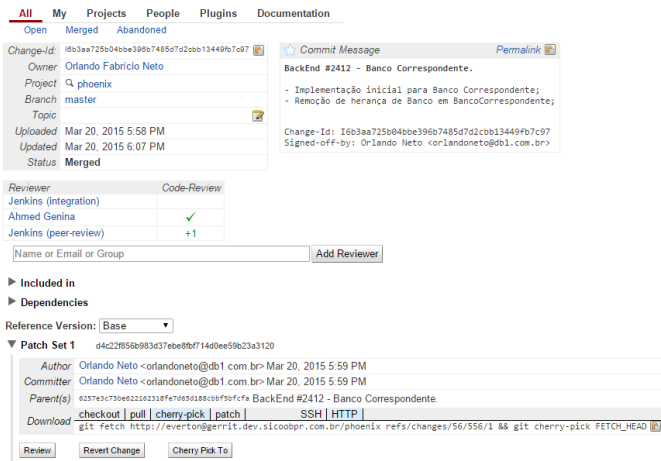


Figure 10. The peer-review authorization.

The Fig. 11 shows the integration of ALM tool (Redmine) with the canonical repository. This allow traceability between requirement, code, peer-review and repository. Among benefits, some of them are: (i) vision end to end (requirement to executable), (ii) fast diagnosis in case of mistake during development (requirement, code, commit, release and other

mistakes) and (iii) integrated management of application life-cycle (requirement, models, code, issues, peer-review, release, incidente, bugs, schedule, documents, wikis, forums, reports, components, library, repository).



Figure 11. The integration of ALM tool with canonical repository.

The Fig. 12 shows a way, using some concepts of TOGAF ³, to developer the requirements (ecosystem, vision, requirement, rules, use case). A fashion used to link an element of Enterprise Architect (EA) and the ALM tool was an ID, put in each one (*GNF001*, for instance).

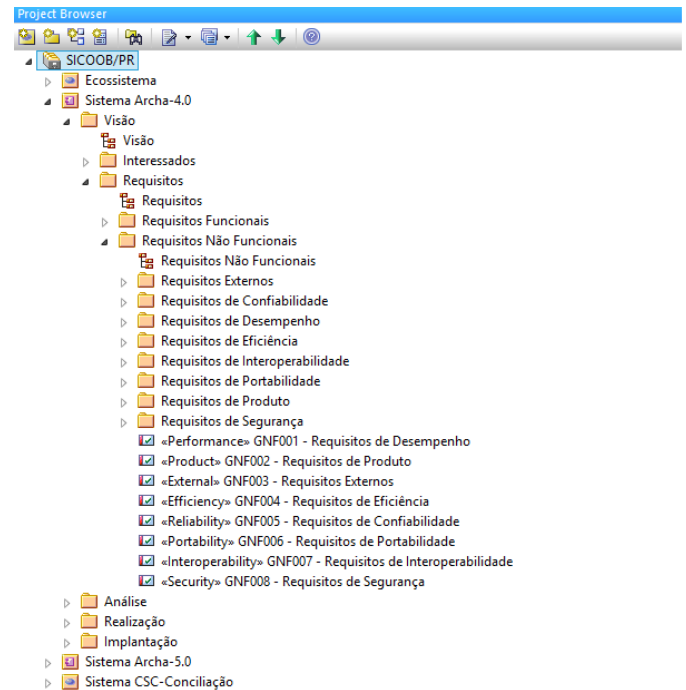


Figure 12. A way to developer requirements using some concepts of TOGAF.

The Fig. 13 shows the IC tool. Besides of all aspects of IC, there are two related with quality, specifically with

²There are some words in portuguese because this picture was collected from a real situation in a brazilian company. The same situation occurs in the figures 10, 12, 13 and 14.

³The Open Group Architecture Framework (TOGAF) is a framework to enterprise architecture which provides an approach for designing, planning, implementing, and governing an enterprise information technology architecture. To see more, www.opengroup.org/togaf

tests: (i) snapshot of automated functional test, using Allure Framework ⁴ and (ii) performance test, allowing identify the performance of application during the builds evolution. The first one is related with audit questions, as create evidence about functional tests and the second one with technical questions, as performance of some elements of application (queries, class, methods, algorithms).

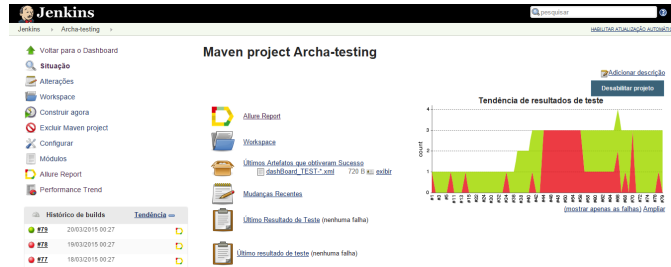


Figure 13. The IC tool.

The Fig. 14 shows the tool used to collect logs from all elements. The importance of this tool is related with diagnosis of all environments: (i) development and (ii) operation. There are a lot of options of tools, for instance, Fluentd ⁵ and Papertrail ⁶. The important is not the tool, but the concept and to get the benefits of use. In case of doubt to choose one, we recommend a method called *Analytic Hierarchy Process* (AHP). More details about its use were shown in [8].

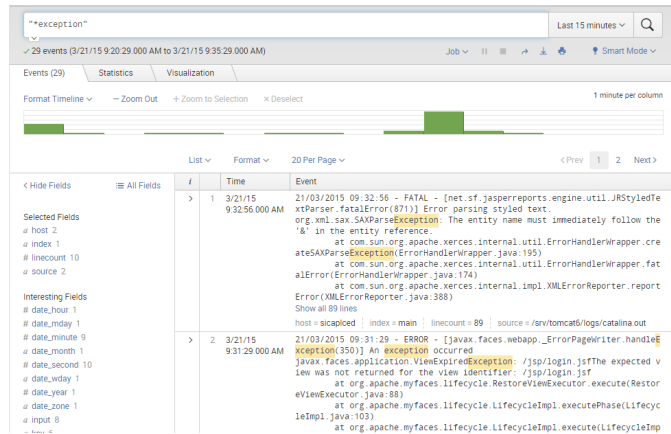


Figure 14. The tool used to collect logs from all elements.

In summary, the servers setup, tools and techniques were used to support the main objective that is put software in operation in a *managed fashion*, reducing risk, rework and increasing traceability, management, predictability and so forth.

As a macro result, the importance of this approach is related with:

- *Traceability and data post-fact*: to allow to link all elements using during software development process (requirement, models, code, versions) and generate a snapshot of all activities from each build (peer-review, tests, deployment).

- *Integrated management*: all team members can use the information generated by ALM tool, like releases, issues, bugs, incidents, documents using only a local.
- *Process standardization*: a standard was created and it is useful to communicate and help team to understand better its process, allowing thus, a possibility of quality increase.

V. CONCLUSION

This work presented a practical approach that can be used in similar processes. Additionally, among the contributions can be mentioned (i) a *set of tools* evaluated and (ii) a *set of techniques*, that can be used for organizations that do not use this type of approach, as for those which already have a higher level of maturity.

Moreover, some further work may be developed to improve setup provided in this article. The first one aims to get a strategy for publication with less impact in terms of unavailability of software products, including deployment across *different timezones*. The second one is linked with *multiple projects* scenarios. We can analyze how the artefacts, from several projects, are promoted to production by the same team.

Finally, this article has a practical purpose. However, to implement continuous delivery and application lifecycle management involves more than installing some tools and automate some tasks. It depends on *effective collaboration* among all of those involved in the delivery process, senior management support and especially the desire of stakeholders in become the changes a reality.

REFERENCES

- [1] M. V. Mantyla and J. Vanhanen, "Software Deployment Activities and Challenges - A Case Study of Four Software Product Companies," 2011 15th European Conference on Software Maintenance and Reengineering, Mar. 2011, pp. 131–140.
- [2] J. Humble and F. David, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. ser. Addison-Wesley Signature Series. Pearson Education, 2010.
- [3] T. Ernawati and D. R. Nugroho, "IT Risk Management Framework Based on ISO 31000:2009," International Conference on System Engineering and Technology, vol. 11, 2012, pp. 1–8.
- [4] B. Fitzgerald, "Continuous Software Engineering and Beyond : Trends and Challenges Categories and Subject Descriptors," RCoSE 14, 2014, pp. 1–9.
- [5] H. Lacheiner and R. Ramler, "Application Lifecycle Management as Infrastructure for Software Process Improvement and Evolution: Experience and Insights from Industry," 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Aug. 2011, pp. 286–293.
- [6] S. Krusche and L. Alperowitz, "Introduction of Continuous Delivery in Multi-Customer Project Courses Categories and Subject Descriptors," ICSE Companion 14, 2014, pp. 335–343.
- [7] S. Bellomo, N. Ernst, R. Nord, and R. Kazman, "Toward Design Decisions to Enable Deployability: Empirical Study of Three Projects Reaching for the Continuous Delivery Holy Grail," 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Jun. 2014, pp. 702–707.
- [8] E. Gomedes and R. M. Barros, "A Non-Intrusive Process to Software Engineering Decision Support focused on increasing the Quality of Software Development," The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27–29, 2013., Jun. 2013, pp. 95–100.

⁴github.com/allure-framework/

⁵fluentd.org

⁶papertrailapp.com

Towards Effective Developer Recommendation in Software Crowdsourcing

Shixiong Zhao, Beijun Shen^{*}, Yuting Chen, Hao Zhong
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China
{649707869, bjshen, chenyt, zhonghao}@sjtu.edu.cn

Abstract—Crowdsourcing has attracted increasing attention from both industry and academia since it was proposed. Now a lot of work is finished by crowdsourcing, such as logo design, website promotion, industrial design, copywriting, software development, translation and image annotation. Although software crowdsourcing achieves positive results in practice, we still face a challenge of assigning suitable developers to specific tasks. In this paper, we propose a novel approach that recommends developers. In particular, our approach supports: comprehensively measuring the tasks and developers in software crowdsourcing, and recommending developers on the basis of the developer-task competence, task-task similarity, and soft power.

Keywords- software crowdsourcing; developer recommendation; developer model

I. INTRODUCTION

Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call [1]. It has attracted great attention from both industry and academia. Now a lot of work is finished by crowdsourcing, such as logo design, website promotion, industrial design, copywriting, software development, translation and image annotation. Meanwhile, when employing crowdsourcing to accomplish software development tasks, we face a challenge of assigning suitable developers to specific tasks. Up to now, most development tasks are assigned in a form of bidding or competition. As a result, much human effort is wasted. Although many attendees would compete for the tasks, some tasks are not well accomplished, since they are not assigned to the most suitable developers.

To the best of our knowledge, most crowdsourcing platforms still rely on crowdsourcers to assign tasks. As a result, crowdsourcers have to spend much time in matching the tasks and the developers. To make things worse, a crowdsourcer is usually biased, since he or she may not have a thoroughly understanding of all developers.

To address the above problem, in this paper, we propose a novel approach that recommends suitable developers to tasks in software crowdsourcing. In particular, our approach uses two models to measure tasks and developers, respectively. For given tasks and developers, our approach computes their

developer-task competences, task-task similarities, and soft powers, and recommends the best candidate(s) to each task.

II. RELATED WORK

Most research on crowdsourcing is concentrated on several topics, including how to apply crowdsourcing [2, 3], how to control the quality of crowdsourcing [4, 5], how to allocate crowdsourcing tasks.

Few researchers discuss how to recommend workers. In particular, I. Boutsis and V. Kalogeraki [6] present REACT that schedules tasks for the crowd under time constraints. It collects worker profiles (*e.g.*, real-time computational capabilities) and dynamically assigns tasks to suitable workers. Research [7] advocates replacing “pull” with “push” for task allocation to achieve higher quality. They analyze the social network of the crowd to gain better performance. E. Simpson and S. Roberts [8] present an information-theoretic approach to assigning workers to specific tasks in crowdsourcing using a Bayesian method. Research [9] proposes to use auctions to map tasks to workers. They organize auctions taking into account price and the suitability of workers estimated based on generated user profiles.

Meanwhile, the existing researches mainly focus on simple tasks, *e.g.*, image annotation, and most researchers just do theoretic research or provide their frameworks. In addition, most of their researches are not domain specific and not complete. They are not able to be applied to software crowdsourcing. We provide an approach to recommend suitable developers to software development tasks.

III. TASK MODEL AND DEVELOPER MODEL

Our approach first abstracts software development tasks and developers with two models. The models measure the tasks and developers quantitatively and thus help establish matching relations between them.

A. Task Model

We model a task as [ID, software category, software size, ability requirement(s), enrolled deadline, task deadline, budget, location].

Each task is associated with one or more ability requirements, and each ability is composed of three sub-attributes: “*type*” and “*name*” are the type and name of an

^{*} corresponding author

TABLE I. DEVELOPER MODEL

Attribute	Sub-attribute	Field
ID	/	/
Base information	Address	/
	Education background	Degree
		Major
	Work experience	Job
Duration		
Service range	/	/
Ability and development experience	Ability	Type
		Name
		Level(1..5)
	Development experience	Date
		Task
		Earning
Evaluation		
Credit and guarantee	Credit	/
	Guarantee	
Task-specific information	Task-ID	
	Bid price	
	Delivery time	

ability requirement; and “*level*” represents the required degree of skills. For example, [*coding language, Java, 3*] indicates that the task requires a Java programmer at a level of “3”.

A task has other attributes: “*software category*” is the kind of the software; “*software size*” is the size of the software, and it could be an estimated value given by the crowdsourcer for the developers to estimate a reasonable delivery time; “*enrolled deadline*” is the deadline for the enrollment; and “*task deadline*” is the deadline to complete the task. Each task is also associated with “*budget*”, working “*location*”.

B. Developer Model

A developer model defines the attributes of developers. As shown in Table I, the developer model has the key attributes of a developer in software crowdsourcing.

“*Base information*” includes personnel information of a developer: “*address*” determines whether a developer can be assigned when a task has the location requirement; “*education background*” and “*work experience*” are strong reference; “*service range*” indicates what kinds of software a developer is able to develop.

The “*ability and development experience*” attribute helps crowdsourcer decide whether or not a developer is competent for a task: “*ability*” is similar to the ability requirement in the task model; “*development experience*” shows the history of a developer in completing software tasks in the crowdsourcing platform.

“*Guarantee*” is the guarantee provided by the developer. It is measured by the security deposit of a developer. “*Credit*” shows the status of development credits: after completing each crowdsourcing task, the developer is evaluated by the crowdsourcer. When calculating the credit of a developer, we consider these factors: if two developers have the same credit value (e.g., four stars), the one with more earnings is preferred (reward-related); the credit changes over time, thus a credit that is gained a long time ago has a weak effect (time-related).

The task-specific information is associated with the task that the developer is enrolled in. It includes task-ID, bid price, and promised delivery time.

IV. DEVELOPER RECOMMENDATION

After a task is released, the crowd can browse and enroll for the task. We recommend the suitable developers among the candidates to tasks. The crowdsourcer selects developers to accomplish tasks according to recommendation list.

Employing our task model and developer model, we calculate the developer-task competence (com_{d-t}), the task-task similarity (sim_{t-t}) and the soft power of developers ($sftpwr$), when recommending developers. In particular, the developer-task competence is used to measure whether a developer is competent for a task; the task-task similarity measures whether a developer has good similar work experience; and soft power reflects personal inner qualities, e.g., the credit of the developer. The three factors complement each other and are useful to match tasks and developers comprehensively. Based on them, we calculate recommendation index ($recindex$) of a developer by (1). The larger $recindex$ is, the higher the developer will be in the recommendation list.

$$recindex = \alpha \times com_{d-t} + \beta \times sim_{t-t} + \gamma \times sftpwr \quad (1)$$

$$\text{where } \alpha + \beta + \gamma = 1.$$

Next, we will introduce how to calculate the developer-task competence, the task-task similarity and the soft power.

A. Developer-task competence

The developer-task competence (com_{d-t}) measures whether a developer satisfies the requirements of a task. We calculate this value by

$$com_{d-t} = com_{dln} \cdot com_{ctg} \cdot (com_{bgt} + com_{loc} + com_{abi}) / 3 \quad (2)$$

It is calculated from the software category competence (com_{ctg}), the ability competence (com_{abi}), the task deadline competence (com_{dln}), the budget competence (com_{bgt}), and location competence (com_{loc}). The requirement of task deadline and category is compulsory, so in (2), com_{dln} and com_{ctg} act as multipliers. The formula of com_{dln} , com_{ctg} , com_{bgt} , com_{loc} , and com_{abi} are defined in Table II.

B. Task-task similarity

If a developer has the development experience of similar tasks in the crowdsourcing platform, it will be an important

TABLE II. CALCULATION OF DEVELOPER-TASK COMPETENCE FACTORS

Factor	Value	Condition
com_{dtn}	1	$delivery\ time \leq task\ deadline$
	0	otherwise
com_{bgt}	1	$bid\ price \leq budget$
	Budget/bid price	otherwise
com_{loc}	1	the developer's address is (in) the required location
	0	otherwise
com_{ctg}	1	the developer's service range cover the required software category
	0	otherwise
com_{abil}	$com_{typ} \cdot com_{nam} \cdot com_{lev}$ calculating one of required abilities of the task	
com_{abi}	$1/n \times \sum_{i=0}^n (com_{abil})_i$ there are n required abilities in the task	
com_{typ}	1	developer has the ability with the same type of the required ability
	0	otherwise
com_{nam}	1	developer has the ability with the same name of the required ability
	0.5	developer only has the ability with the name under the required type e.g., name of required ability: C, name of developer's ability: C++
	0	otherwise
com_{lev}	1	$level_w \leq level_r$
	$level_w / level_r$	otherwise

reference. As a result, we take the task-task similarity into account, and here defines a task as $T = [cat, typ, nam, lev, bgt]$. The i -th done task of a developer is presented with $T_i = [cat_i, typ_i, nam_i, lev_i, bgt_i, evl_i]$, where Cat , typ , nam , lev and bgt represent category, ability type, ability name, ability level, and budget, respectively; and evl is the evaluation of a developer gotten from the crowdsourcer on this task.

We use Soft Jaccard's Coefficient to measure the task-task similarity. Suppose that the evaluation value is an integer from 1 to 5, and the developer has n finished tasks in total, sim_{t-t} is calculated as follows:

$$sim_{t-t} = \max \left\{ \frac{|T \cap T_i|}{|T \cup T_i|} \cdot \frac{evl_i}{5} \right\} (i = 1, \dots, n) \quad (3)$$

For cat , if $cat = cat_i$, they are regarded as intersected. Otherwise they are disjoint. So are typ and nam . For lev , if the name of the i -th finished task is same with the task, and lev_i is not lower than lev , they are regarded as intersected. Otherwise, they are disjoint. For bgt , if $1/5 \leq bgt/bgt_i \leq 5$, they are regarded as intersected. Otherwise, they are disjoint. When judging whether they are intersected or not, the condition is relaxed but not just depending on whether they are the same, so we call it Soft Jaccard's Coefficient.

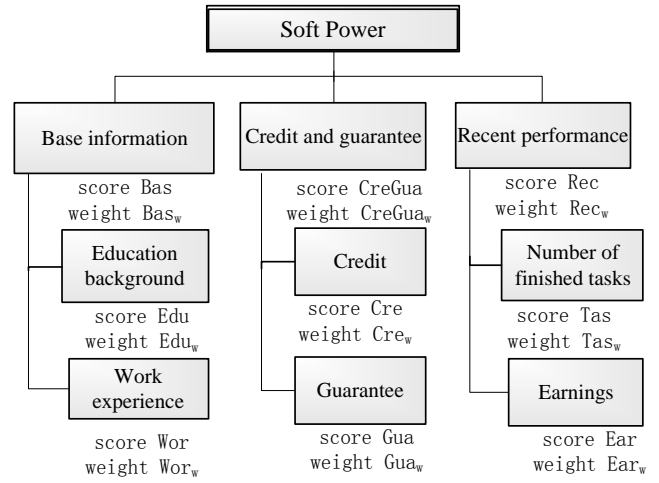


Figure 1. Composition of soft power

Although a developer has experience on a similar task, she/he may not do the task well. Considering this issue, when calculating the task-task similarity, our approach introduces the crowdsourcer's evaluation, evl_i . All the similar tasks of a developer are calculated one by one, and we select the maximum value as sim_{t-t} .

C. Soft power

Some information of a developer are not reflected in com_{d-t} and sim_{t-t} , e.g., the education background, work experience (not in software crowdsourcing), the credit, the guarantee of a developer, and the recent performance. We call these factors soft power. The composition of soft power is in Fig. 1. " $sftpwr$ " is calculated by

$$sftpwr = Bas \times Bas_w + CreGua \times CreGua_w + Rec \times Rec_w \quad (4)$$

where $Bas_w + CreGua_w + Rec_w = 1$

Now we will focus on "Bas", "CreGua" and "Rec".

1) "Bas"

The base information includes education background and work experience. The original " Bas_0 " is calculated by (5). Suppose that $BASE$ is the maximum value among all the enrolled developers, we divide " Bas_0 " by $BASE$ to get "Bas".

$$Bas_0 = Edu \times Edu_w + Wor \times Wor_w \quad (5)$$

where $Edu_w + Wor_w = 1$

The education background includes some [degree, major] tuples. "Edu" is calculated by (6). The work experience includes some [job, duration] tuples. "Wor" is calculated by (7). The score of degree ($Degree$) is 1, 2, 3 for bachelor, master, doctor respectively, and 0 for others. The score of major ($Major$) is 1 if the major is software-related, 0 otherwise, and so is the score of Job (Job). We divide the duration into several intervals, and the score of duration ($Duration$) is 1 if the

developer's work duration is in $[0, dura_1)$, 2 if it is in $[dura_1, dura_2)$, and so on.

$$Edu = \sum_{i=1}^n Degree_i \times Major_i \quad (6)$$

$$Wor = \sum_{i=1}^n Job_i \times Duration_i \quad (7)$$

2) "CreGua"

"CreGua" includes credit and guarantee. The original "CreGua₀" is calculated by (8). Suppose that CREGUA is the maximum value among all the enrolled developers, we divide "CreGua₀" by CREGUA to get "CreGua".

$$CreGua_0 = Cre \times Cre_w + Gua \times Gua_w \quad (8)$$

where $Cre_w + Gua_w = 1$

"Cre" consists of two parts, good record and bad record. The weight of bad record (Bad_w) is greater than weight of good record ($Good_w$), because we punish those developers who have "bad" record, since employers would not deliver tasks to developers with bad credits. As the credit is time-related and reward-related, we introduce two parameters, $Time_w$ and $Reward_w$. $Time_w$ is calculated by (9), where T represents the months of age of the crowdsourcing platform, and Δt represents the period (months) for the evaluation. This formula ensures that the longer from now, the less effect the credit record has. The effect won't disappear. $Reward_w$ is calculated using the similar method as "Duration" but the intervals are divided by reward. Since there can be many credit records, "Cre" of a developer is calculated by (10).

$$Time_w = (T - \Delta t) / T \times e^{-\Delta t / T} \quad (9)$$

$$Cre = \sum_{i=1}^n Good_w \times Time_{wi} \times Reward_{wi} - \sum_{i=1}^m Bad_w \times Time_{wi} \times Reward_{wi} \quad (10)$$

"Gua" presents the guarantee of a developer. It is measured by the security deposit of a developer. "Gua" is calculated using the similar method as "Duration" but the intervals are divided by security deposit.

3) "Rec"

The recent performance often reflects the activeness of a developer. The more active a developer is, the more effort can be put in crowdsourcing. "Rec" is calculated by combing the number of finished tasks ("Tas") of the developer, and earnings he/she has made ("Ear") in the recent 3 months. The original "Rec₀" is calculated by (11), where "Tas" and "Ear" are calculated using the similar method as "Duration" but the intervals are divided by number of recent finished tasks and recent earnings, respectively. Suppose that REC is the

maximum value of all enrolled developers, we divide "Rec₀" by REC to get "Rec".

$$Rec_0 = Tas \times Tas_w + Ear \times Ear_w \quad (11)$$

where $Tas_w + Ear_w = 1$

V. EVALUATION PLAN

In the future, we will evaluate our approach in the three steps:

- Data preparation. We will fetch the most suitable data of accomplished software development tasks and do preprocessing.
- Parameter tuning. We will use greedy search to find the best value of the parameters in our approach.
- Precision comparison. We will recommend developers to tasks with our approach and compare the precision of our approach with a general approach.

VI. CONCLUSIONS

There exists a challenge of assigning the most suitable developers to specific tasks in software crowdsourcing. In this paper, we proposed an approach of recommending developers for software crowdsourcing. Firstly, our approach models the task and the worker comprehensively. Then our approach recommends developers for software crowdsourcing tasks based on their developer-task competence, task-task similarities and developer's soft powers.

ACKNOWLEDGMENT

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242, 91118004).

REFERENCES

- [1] J. Howe, "The rise of crowdsourcing," *Wired*, pp. 1-4, June 2006.
- [2] A. Brew, D. Greene, and P. Cunningham, "Using crowdsourcing and active learning to track sentiment in online media," in *Proc. 19th European Conf. on Artificial Intelligence*. Lisbon, 2010, pp. 145-150.
- [3] O. F. Zaidan and C. Callison-Burch, "Crowdsourcing translation: Professional quality from non-professionals," in *Proc. 49th Annual Meeting of the Association for Computational Linguistics*. Portland, 2011, pp. 1220-1229.
- [4] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H.R. Motahari-Nezhad, E. Bertino, and S. Dustdar, "Quality Control in Crowdsourcing Systems: Issues and Directions," *IEEE Internet Computing*, Vol. 17, pp. 76-81, March-April 2013.
- [5] U. Hassan and E. Curry, "A Capability Requirements Approach for Predicting Worker Performance in Crowdsourcing," in *9th Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing*. Austin, 2013, pp. 429-437.
- [6] I. Boutsis and V. Kalogeraki, "Crowdsourcing under Real-Time Constraint," in *IEEE 27th Int. Symp. on Parallel & Distributed Processing*. Boston, 2013, pp. 753-764.
- [7] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux, "Pick-A-Crowd: tell me what you like, and I'll tell you what to do," in *WWW 2013*. Rio de Janeiro, 2013, pp. 367-374.
- [8] E. Simpson and S. Roberts, "Bayesian Methods for Intelligent Task Assignment in Crowdsourcing Systems," in *Studies in Computational Intelligence*, Springer, vol. 538, pp 1-32, February 2015.
- [9] B. Satzger, H. Psai, D. Schall, and S. Dustdar, "Auction-based crowdsourcing supporting skill management," in *Information Systems*, vol.38, pp. 547-560, June 2013.

Creating User Scenarios through User Interaction Diagrams by Non-Technical Customers

Douglas Hiura Longo and Patrícia Vilain
Informatics and Statistics Department,
Federal University of Santa Catarina,
Florianopolis, Brazil
douglasshiura@inf.ufsc.br, patricia.vilain@ufsc.br

Abstract—This paper investigates the applicability of User Interaction Diagrams (UIDs) as user scenarios for specifying requirements of software built by non-technical customers. User scenarios represent an alternative to representation of Acceptance Test-Driven Development (ATDD). Two methods for building user scenarios using UIDs were proposed: the progressive and the regressive methods. The progressive method for construction of scenarios provides a description from any starting point until the expected result is reached. The regressive method is based on the Assert-First technique, introduced in Test-Driven Development (TDD), where the user scenario is constructed the other way round, that is, from the expected result to the starting point. These two methods were applied in an experiment where the results demonstrated that the regressive method requires significantly less effort as compared to the progressive method. The quality criteria of the two methods were different, where the regressive method yielded better results.

Keywords: *Requirements Engineering; ATDD; Assert First; TDD; User Scenarios.*

I. INTRODUCTION

It is known that understanding user requirements is critical to the success of a project [1]. The basic idea of Automated Acceptance Testing - AAT is to document requirements and desired outcomes in a format that can be automatically and repeatedly tested [2]. AAT represents customer expectations [1] and was adopted in agile software development holding great promise of improving communication and collaboration among those involved [2, 3, 4].

According to Hoffmann et al., Acceptance Test-Driven Development - ATDD facilitates the requirements specification, raising awareness, to those involved, of the importance of testing as an auxiliary mechanism for quality assurance. In theory, the customer expresses requirements as input to the software paired with some expected result [5]. However, in practice, customers prefer to express requirements at interaction meetings, while acceptance tests are written by developers [2].

Alvestad investigated whether a non-technical customer could express requirements based on domain specific languages, but was not successful in confirming his assumptions [6]. Domain specific languages are used by tools for automated tests. The tools for AAT support these

requirements representations: formal, semi-formal and informal (NL, Stories, Tables). However, such tools do not include the end user in the requirements specification process [7]. According to Haugset and Hanssen, the application of AATs is barely reflected in practice and it is somewhat inappropriate for customers to express requirements in the form of automated acceptance tests [2].

The tools or languages in general induce the sequential creation of a requirement specification, which is where the requirement is specified, starting from any point towards the desired outcome. This is called the progressive method.

However, Test-Driven Development - TDD is a set of development practices, where the code is developed from tests. The Assert-First technique has a powerful simplifying effect during test development [8]. This technique consists of writing the test assertions first by following a regressive process in order to complete the test, writing the minimum of code lines. In this way, the regressive method consists of creating the requirement specification from the result.

User Interaction Diagrams (UIDs) represent the interaction between the user and the system, and can support the users' representation of scenarios [9, 10]. The present study investigates the possibility of a non-technical customer to express the system requirements by using UIDs as user scenarios. However, it also investigates the creation of requirements specification using the progressive and regressive methods.

To evaluate the proposal, we considered an experiment with 21 non-technical participants, and the requirements specification for a game. The objective of the experiment is to demonstrate the use of UIDs as an agile method for requirements specification, and check different results between the two methods.

This paper is organized as follows: Section II presents details of the proposal. Section III describes the evaluation methodology for verifying the proposal's efficacy. The results of the study are presented in Section IV. Section V presents threats to validity. Finally, the conclusions are presented.

II. RESEARCH PROPOSAL

This study uses UIDs for representing software requirements. This proposal replaces the information types

represented in the UIDs by values of the user scenarios. Fig. 2 shows an example of user interaction with a calculator and the representation of a user scenario through a UID.

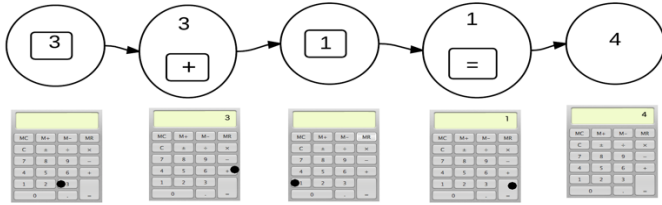


Figure 1. User scenario of a calculator for the sum function, with sample pictures of the user's interaction (points) with the calculator.

Fig. 1 shows the user's interaction with the calculator, following the user scenario. In this example, the user enters the values for the sum ($3 + 1 =$) and the system displays the result (4).

Table 1 below presents the symbols for the language to represent user scenarios through UIDs.

TABLE I. SYMBOLS FOR THE LANGUAGE OF USER SCENARIOS REPRESENTATION THROUGH UIDs.

Symbol	Use
	Ellipse - represents a state of interaction
	Arrowed line - represents the transition between interaction states and flow direction.
	Rectangle - represents the user input, its value is represented by a set of characters contained within the ellipse.
Characters sequence	Value - represents the system output, where a set of characters is contained within the ellipse.

Every interaction state (ellipse) contains the values of the user input and system output. The flow of the interaction states is represented by the direction of the transition. The initial state is the first interaction state following the direction of the arrows. The final state, in turn, is the last interaction state of the flow. For the construction of the requirements specification as user scenarios, the following methods are presented:

- **Progressive:** it indicates the expected result only at the end of the construction flow. Initially, the interaction states are built in order to achieve the expected result. Fig. 1 shows the progressive flow of the construction of a requirement specification, that is, every specification of the sum function is constructed, and its result is shown only at the end of the flow.
- **Regressive:** similarly to the Assert-First technique [8], the regressive method starts the construction of the user scenario from the result, and adds other interaction states specifically to reach the outcome (initial state). Fig. 2 shows the scenario where the requirement is constructed with the regressive method.

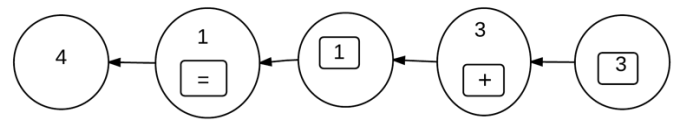


Figure 2. Regressive method to create the sum function scenario of a calculator.

III. ASSESSMENT OF THE PROPOSAL

The purpose of user scenarios is the specification of software requirements. Customers and developers to promote communication and collaboration can use these scenarios during development. However, it is important that such requirements be complete, consistent and realistic [11, 12].

To assess applicability and usefulness of the proposal in relation to completeness and consistency of requirements represented by non-technical¹ participants, we conducted an experiment. The experiment aimed at the construction of user scenarios of the 8-Puzzle game. We proposed to investigate the following questions:

RQ1: What quality factors (completeness and correctness) of the requirements are represented in user scenarios?

RQ2: Are such quality factors (completeness and correctness) associated with progressive or regressive methods?

RQ3: Which of the proposed methods facilitates the construction of user scenarios?

A. The 8-Puzzle Game

8-Puzzle is a game consisting of a board with 3 rows and 3 columns. The board has a number sequence from 1 to 8 and an empty space. The goal is, starting from a random state, to order the sequence of numbers. Fig. 3 shows the final state.

1	2	3
4	5	6
7	8	

Figure 3. Final state of the 8-Puzzle game.

The operations for the configuration of the board include moving the empty space up, left, right or down. In digital implementations, the empty space is moved by using the keyboard arrow keys or by clicking on a number next to the empty space.

B. Methodology for assessment

For the assessment, we considered the construction of user scenarios for the requirements specification of the 8-Puzzle game with non-technical participants. The materials needed were: a pencil or pen, an eraser, and blank paper. Fig. 4 shows the diagram with the activities carried out during the three evaluation stages: preparation, experiment and result analysis.

¹ Non-Technical participants are not knowledgeable about UIDs, FIT, and Automated Acceptance Testing.

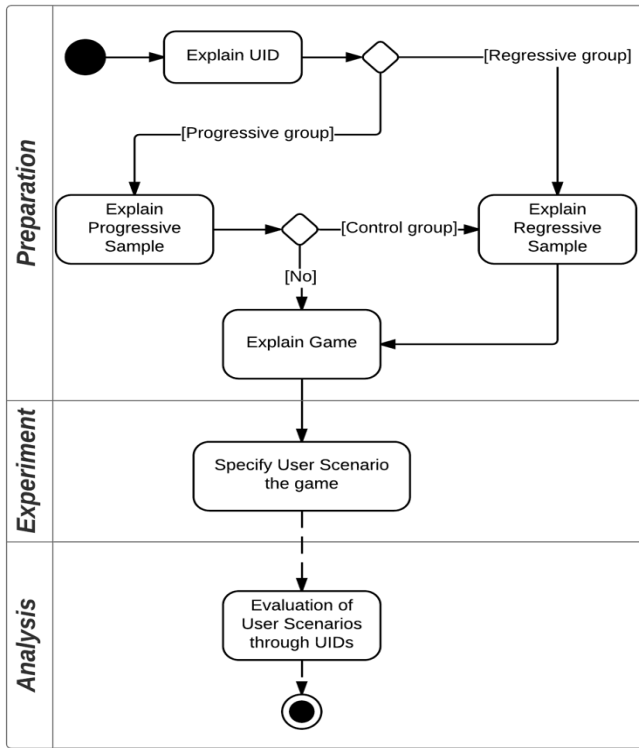


Figure 4. Diagram of activity for assessment of user scenarios through UID.

1) *Preparation*: In order to answer the questions and analyze the differences between the methods, the participants were divided into three groups: progressive, regressive and control (progressive/regressive). In the preparation stage, the participants were trained for about 15 minutes, according to each group. So, during the preparation stage of the progressive group, explanation activities were carried out about: UIDs, the progressive method, and the 8-Puzzle game. The regressive group carried out explanation activities about: UIDs, the regressive method, and the 8-Puzzle game. For the control group, explanation activities were performed about: UIDs, the progressive and regressive methods, and the 8-Puzzle game. This way, the participants of the control group made their own choice of method during the experiment. During the explanations, the participants were allowed to use a pencil and some paper in case they wanted some practice.

2) *Experiment*: In the experiment stage, each participant had to specify the final state of the 8-Puzzle game, or victory, still considering some user interaction for the board configuration. The user scenarios had to be developed using paper and a pencil or pen, at the participant's choice. Each participant had the maximum time limit of 15 minutes for the experiment performance. The time spent by each participant to complete the task was collected during the experiment.

C. Analysis of question RQ1

Fig. 6 shows an expected user scenario of the 8-Puzzle game.

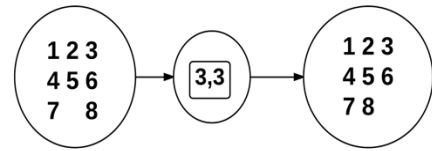


Figure 5. Expected user scenario of the 8-Puzzle game.

User scenario variations such as user input to move the pieces (numbers) on the board, quantity of interactions, and direction of transition flows were considered adequate even being different from the diagram in Fig. 5. Question RQ1 can be answered by evaluating the user scenarios. The analysis considered the evaluation of the quality factors: completeness and correctness.

1) *Completeness*: It means that all user-required services must be defined [11, 12]. Completeness can be evaluated by assigning complete/incomplete values. It is considered to be complete the user scenario that presents:

- The end result or state of victory; and
- At least one user input to the board configuration.

In [13], incompleteness is defined as ambiguity type. It occurs when a statement fails to provide enough information to have a single clear interpretation. It is considered to be incomplete the user scenario that:

- Does not present state of victory;
- Does not present interaction of playing; or
- Is incorrect.

2) *Correctness*: It is the quality factor indicating whether the participant understands and correctly applies the UID language. Correct/incorrect values are assigned for this quality factor. The correct value is assigned to the user scenarios where the language symbols are properly applied in accordance with Table I. The incorrect value is assigned to the user scenario that:

- Contains cyclic transitions;
- Contains transitions to more than one interaction state;
- Contains transitions to random values; or
- Does not consider the flow of transitions.

D. Analysis of Question RQ2

Question RQ2 can be answered by the formula below with the following hypothesis:

$$H_0 : F_{\text{Progressive}} = F_{\text{Regressive}}$$

And (1)

$$H_1 : F_{\text{Progressive}} \neq F_{\text{Regressive}}$$

- $F_{\text{Progressive}}$ is completeness and correctness of the user scenarios progressively specified; and

- $F_{\text{Regressive}}$ is completeness and correctness of user scenarios regressively specified.

The decision to accept H_0 or H_1 is made from the data collected from the experiment. By accepting the H_0 hypothesis, it can be stated that the quality factors of the expressed requirements are indifferent, that is, these quality factors are not associated with the method. However, by accepting the H_1 hypothesis, we affirm that the quality factors are different between the two methods, that is, these quality factors are associated with the method. The statistical test significance level should be at least 5% ($\alpha = 0.05$).

E. Analysis of Question QR3

The easiness or effort to construct the user scenarios are analyzed through time data. Therefore, it is necessary to carry out a distribution analysis of the time spent by the participants according to the method used for comparison.

IV. RESULTS

A. Preparation Stage

The experiment was conducted with 21 participants. The participants were prepared according to the progressive and/or regressive methods, resulting in three groups. The progressive group consisted of 8 participants. The regressive group was composed of 6 participants. The control group consisted of 7 participants. The graph in Fig. 6 shows the education level of the participants.

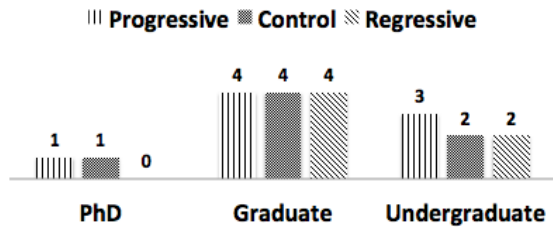


Figure 6. Education level of the participants.

The participants' ages range from 22 to 45 years. Fig. 7 shows box plots with the participants' age variation in each group.

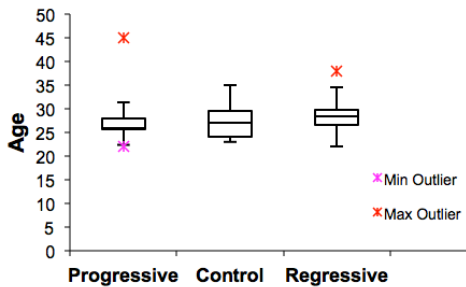


Figure 7. Participants' age variation in each group.

B. Experiment results

The user scenarios delivered by the participants were initially classified according to the progressive or regressive methods. Ten participants used the progressive method and eleven participants used the regressive method. The choices of

the control group were 29% progressive method and 71% regressive method. Table II presents the correlation matrix between methods used during preparation and methods used by the participants during the experiment.

TABLE II. CORRELATION MATRIX BETWEEN PREPARATION AND EXPERIMENT.

Correlation Matrix		Experiment	
		Progressive	Regressive
Preparation	Progressive	8	0
	Regressive	0	6
	Control	2	5

The quality factors (completeness and correctness) of user scenarios are evaluated according to Section IV (Analysis of question RQ1).

1) *Complete and Correct*: The assessment considered fourteen of the user scenarios as complete and correct. As an example, Fig. 8 shows a user scenario that applied the progressive method, which was assessed as correct and complete.

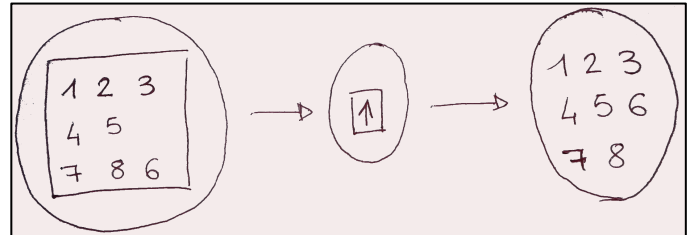


Figure 8. Complete and correct user scenario using the progressive method.

Fig. 9 and 10 show two scenarios that applied the regressive method and were assessed as correct and complete.

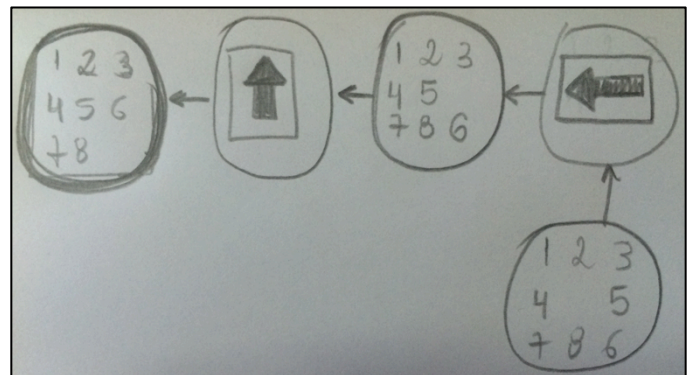


Figure 9. Complete and correct user scenario using the regressive method, applying movement to the number adjacent to the empty space.

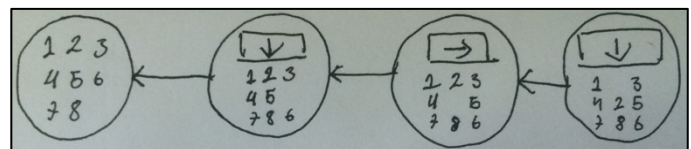


Figure 10. Complete and correct user scenario using the regressive method, applying movement to the empty space.

2) *Incomplete and Correct*: The assessment considered five of the user scenarios as incomplete and correct.

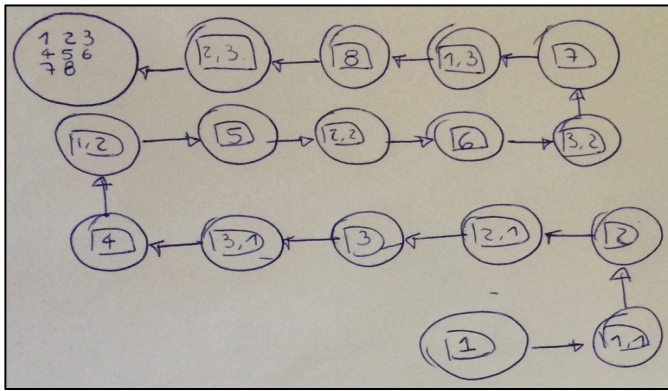


Figure 11. Incomplete and correct user scenario using the regressive method, not specifying interaction of playing.

3) *Incorrect and Incomplete*: In only two cases, the participants used UID incorrectly. In such cases, it is assumed that the requirement was incomplete due to absence of syntax.

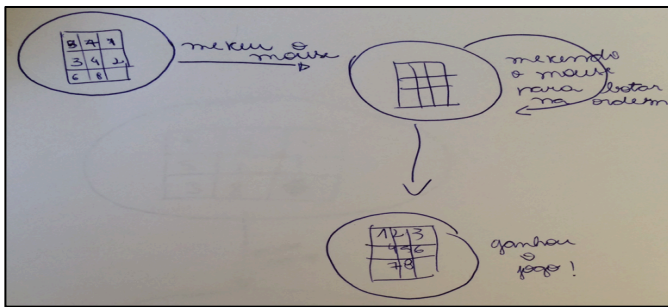


Figure 12. Incomplete and incorrect user scenario using the progressive method, an unintelligible user scenario.

In an overall assessment result, the participants delivered 67% complete user scenarios where 90% of them used UIDs correctly.

The graph in Fig. 13 displays the completeness and correctness distribution divided according to the scenario method of construction.

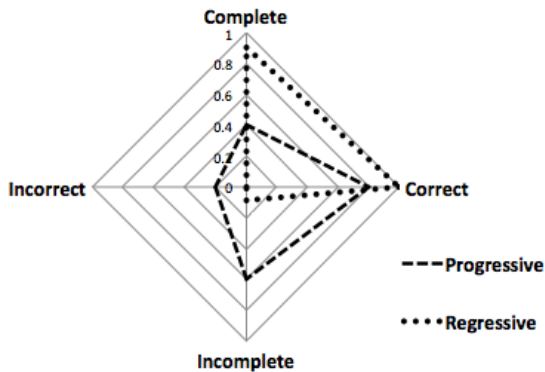


Figure 13. Graph showing the probability of correctness and completeness, divided according to construction method.

Axis "x" on the graph in Fig. 13 represents the distribution of correctness and axis "y" represents completeness. The positive area of axes "x" and "y" represents the best quality factor. It can be seen that the user scenarios specified by the progressive method have a 40% probability of being complete, and 80% of being correct. And the regressive method has 91% probability of being complete, and 100% of being correct (RQ1).

To demonstrate the difference of quality factors between the methods, we used Fisher's statistical test [14].

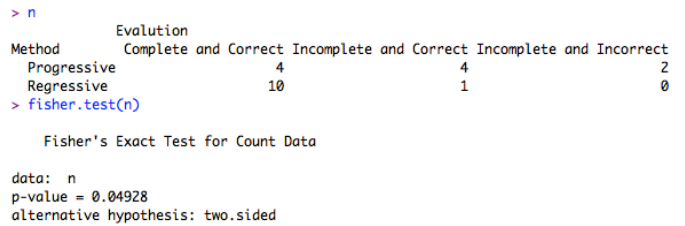


Figure 14. Fisher's statistical test applied with the statistical tool R.

According to Fisher's statistical test, p-value is 0.04928. However, the confidence level of the test must be ($\alpha = 0.05$). Thus, $p - value < \alpha$ then the alternative hypothesis (H_1) is considered, allowing to conclude that the quality factors are different between the methods (RQ2).

The box plot in Fig. 15 shows the distribution of the time spent divided according to the construction method.

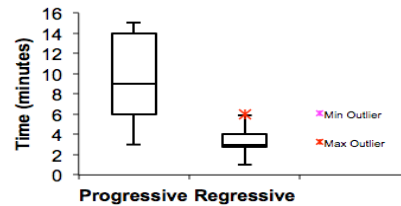


Figure 15. Distribution of time spent, according to construction method.

The time spent by the progressive group is between 3 and 15 minutes, and the regressive group is between 1 and 6 minutes. The regressive method has an outlier. The outlier (6 minutes), out of the distribution of time spent by the group, indicates that a participant may have found it more difficult to specify and develop the user scenario. The average time used by the progressive and regressive groups is 9 minutes and 3 minutes, respectively. Thus, according to the time distribution analysis, the regressive method involves less effort (RQ3).

V. THREATS TO VALIDITY

While the 8-puzzle is simple and effective for experimental purposes, abstracting from this game scenario to the use of UID in real-world software project requirements is not easy. The case study research is incomplete without a discussion of concerns that may threaten results validity. Internal validity refers to the causal inferences made based on experimental data [15]. Our goal is simply to determine whether and how participants create user scenarios through UIDs.

The case study has two important considerations: knowledge of the case study; creation process of user scenarios.

The knowledge of the case study by participants refers to the domain on the 8-puzzle, considering the logic and rules. This knowledge is common to the participants. The user scenarios creation process is different from the process of reproduction (copy) of user scenarios. An example of reproduction is to copy the calculator user scenario (Fig. 1) and simply change the user setting values. Our case study avoids the process of reproduction of the participants, considering only the creation of user scenarios. We also observed that the problem domain allowed the evaluation of the effort to create user scenarios without interruption for fatigue.

Construct validity refers to the appropriate use of evaluation metrics and measures [15]. We specifically avoid the use of absolute measures of completeness and correctness conforms the term as expressed in accepted IEEE standards [16]. To calculate other statistical measures, we used accepted statistics for agreement (Fisher's Exact Test) and scrupulously followed recommended practices in applying them.

External validity refers to the ability to generalize the findings to other domains [15]. The external validity of research contains two threats: the problem domain studied and the population of participants. The problem domain (8-puzzle) contains user interactions with the system. These user interactions are similar, such as: a calculator; authentication systems; or in areas of other studies that consider the applicability of UIDs in software engineering, as in [9] [10] [17].

Unfortunately, the study used a small population of participants, rather than a large population of participants. However, the population of participants was composed of different educational levels, and related areas of business, engineering and science.

Reliability refers to the ability of other researchers to replicate methodology [13]. We detail the proposal and the evaluation technique and the result, and we consider important other researchers to reproduce our study.

VI. CONCLUSIONS

The challenge of eliciting requirements from customers is worthy of investigation and so is any effort to simplify or assist in the process. The paper does both and some interesting results are shared. The applicability of user scenarios in the present study is related to agile software development, where requirements are customer expectations and should as well be used as tests for the application code. UIDs were used to allow non-technical customers to represent user scenarios. In this context, UIDs were quite suitable for creating user scenarios to specify software requirements.

As for the assessment of this proposal, an experiment was conducted with 21 non-technical participants to specify the requirements of a game. With statistical analysis of the experiment results, it was observed that the progressive and regressive specification methods are different. The regressive method resulted in 91% of complete requirements while the progressive method resulted in 40% of complete requirements. The participants delivered 67% complete user scenarios where 90% of them used UIDs correctly. However, in our study case, the novelty of the proposed regressive method based on the

TDD assert-first technique is the reduction of effort, and improvement in the assessed quality factors of the requirements.

In spite of the fact that this study has considered only UIDs for representing user scenarios, a tool for automated acceptance tests is being built, with support for direct execution of user scenarios.

REFERENCES

- [1] R. Miller and C. T. Collins, "Acceptance testing," Proc. XPU Universe. 2001.
- [2] B. Haugset and G. K. Hanssen, "Automated acceptance testing: A literature review and an industrial case study," Agile, 2008. AGILE'08. Conference, IEEE. 2008, pp. 27-38.
- [3] T. Dybå and T. Dingsøy, "Empirical studies of agile software development: A systematic review," Information and software technology. Elsevier, vol. 50, 2008, pp. 833-859.
- [4] G. K. Hanssen and B. Haugset, "Automated acceptance testing using fit," System Sciences, HICSS'09, 42nd Hawaii International Conference on. IEEE, 2009, pp. 1-8.
- [5] L. F. S. Hoffmann, L. E. G. D. Vasconcelos, E. Lamas, A. M. D. Cunha and L. A. V. Dias, "Applying Acceptance Test Driven Development to a Problem Based Learning Academic Real-Time System," Information Technology: New Generations (ITNG), IEEE, 11th International Conference on. 2014, pp. 3-8.
- [6] K. Alvestad, "Domain Specific Languages for Executables Specifications," Institutt for datateknikk og informasjonsvitenskap. p. 63, 2007.
- [7] M. Kamalrudin, S. Sidek, M. N. Aiza, and M. Robinson, "Automated Acceptance Testing Tools Evaluation in Agile Software Development," Sci. Int. 2013, pp. 1053-1058.
- [8] K. Beck, Test Driven Development: By Example. Addison-Wesley Professional, 2003.
- [9] N. Güell, D. Schwabe and P. Vilain, "Modeling interactions and navigation in web applications," Conceptual Modeling for E-Business and the Web. Springer, 2000, pp. 115-127.
- [10] P. Valderas, V. Pelechano, "A survey of requirements specification in model-driven development of web applications," ACM Transactions on the Web (TWEB). ACM, Vol. 5, p.10, 2011.
- [11] I. Sommerville, Software Engineering. 9th ed, p. 773. Addison-Wesley, 2011.
- [12] A. D. Lucia and A. Qusef, "Requirements engineering in agile software development," Journal of Emerging Technologies in Web Intelligence. vol. 2, 2010, pp. 212-220.
- [13] A. K. Massey, R. L. Rutledge, A. I. Anton, P. P. Swire, "Identifying and classifying ambiguity for regulatory requirements," Requirements Engineering Conference (RE), 2014 IEEE 22nd International, IEEE, 2014, pp. 83-92.
- [14] E. L. Lehmann, and J. P. Romano, "Testing statistical hypotheses". Springer, 2006.
- [15] R. K. Yin, *Case Study Research: Design and Methods*, 3rd ed., ser. Applied Social Research Methods Series, L. Bickman and D. J. Rog, Eds. Sage Publications, 2003, vol. 5.
- [16] IEEE Recommended Practice for Software Requirements Specifications," IEEE Std 830-1998 , pp.1-40, 1998.
- [17] N. V. Zeferino and P. Vilain, "A model-driven approach for generating interfaces from user interaction diagrams," Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services. ACM, 2014, pp.474-478.

How Stakeholders' Commitment May Affect the Success of Requirements Elicitation

Corentin Burnay*^{†‡}, Ivan Jureta*^{†‡} and Stéphane Faulkner^{†‡}

*Fonds de la Recherche Scientifique – FNRS, Brussels

[†]Department of Business Administration, University of Namur

[‡]PReCISE Research Center, University of Namur

Abstract—Requirements elicitation consists in collecting information about the requirements and the environment of a system-to-be. It usually involves business analysts who are eliciting information, and stakeholders who are providing information. This paper investigates how the commitment of stakeholders to a RE project influences the results of elicitation. We suggest a way to measure the commitment of stakeholders during RE, and propose the so-called “commitment matrix”, which shows what analysts can expect from stakeholders who are more, as opposed to those who are less committed. The matrix builds on a survey of 87 stakeholders. Our results suggest that commitment somehow affects the information provided by stakeholders, and that it is therefore a relevant criterion to account for when selecting stakeholders to be involved in elicitation.

Keywords—Requirements Elicitation, Commitment, Involvement

I. INTRODUCTION

1) *Context*: Requirements Engineering (RE) focuses on the elicitation, representation, and analysis of requirements and environment of a system-to-be, in order to produce its specification. The specification should be such that, if the system-to-be satisfies the specification, then it will also satisfy its requirements within its environment. RE involves various tasks, such as the elicitation of requirements, their representation, the analysis of their consistency, their validation and negotiation with the system stakeholders, and so on. We focus in this paper on the elicitation aspect; that is, on the acquisition of information about the expectations of stakeholders toward the system-to-be, and about the environment in which that system will run. We will say that this information is represented in *elicitation documentation*, such as interview transcripts or recordings, notes from field observation, questionnaires and similar. There are at least three recurring challenges that should be considered when preparing the elicitation documentation:

- *Documentation Quality*: information must be easily understandable, and should be sufficiently stable so that analysts can actually rely on the information [1], [2];
- *Documentation Quantity*: information must cover the relevant requirements of the stakeholders, and should not overlook important aspect of the environment, so as to minimize requirements incompleteness [1], [2], [3];
- *Documentation Efficiency*: information must be obtained at reasonable cost and resource use, i.e., it is important to achieve quality and quantity, but not at any cost [4].

In the rest of this paper, we refer to these three Requirements Elicitation Challenges as RECs. How successfully one deals with REC is influenced, among other things, by the choice of some elicitation techniques to apply [5] and the selection of stakeholders to apply them with [6], [7].

2) *Problem and Research Question*: Identifying stakeholders is hardly a new issue in RE [8]; most of the time, a person is considered as a stakeholder whenever she has a stake in the system [9]. In RE, it has been suggested that a stakeholder is any person or organization who influences a system's requirements or who is impacted by that system [10]. Such definition likely leads to a large set of stakeholders, which can hardly be entirely solicited. The question of selecting stakeholders to be involved from the pool of available stakeholders is, to the best of our knowledge, a topic that has received smaller attention from RE community. More precisely, we believe that one possible way of selecting some stakeholders is to account for their respective *level of commitment to the RE project*, i.e., how the stakeholders are intellectually or emotionally bounded to the RE project. For instance, involving two stakeholders with different commitment levels during elicitation could result in elicitation documents that differ in terms of quality, quantity or efficiency. In fact, the committed stakeholder may be cautious and mention any piece of information she has, so that elicited quantity increases. On the other hand, the same stakeholder may be too cautious and share irrelevant information, thereby decreasing the quality of the information. In this paper, we therefore investigate the question of what changes whenever a stakeholder is committed or not to a RE project. We do so for the three main RECs, i.e., does commitment influence quality, quantity and/or efficiency of the elicitation documentation. Ultimately, we believe such contribution could help business analysts to select more accurately stakeholders so as to improve the chances of success for the system-to-be.

3) *Contributions*: Our contribution in this paper is twofold. Firstly, we discuss and clarify the distinction between the concept of involvement and commitment in elicitation, and we describe reasons why commitment may be an important criterion to account for when selecting stakeholders. Secondly, we survey 87 stakeholders in order to better understand how stakeholders' commitment to a RE project may influence the quality, quantity and efficiency of the elicitation documentation. Based on these results, we propose a commitment matrix, showing the relative advantage of involving stakeholders with various commitment levels. The matrix is shown in Figure 1 and reads as follows; if a business analyst needs to collect information that is, for example, clear, then she can involve any

Requirements Elicitation Challenges (RECs)			
	Quality	Quantity	Efficiency
Level of Stakeholder's Commitment	Strong Feasible: stakeholders share requests about the future system that they believe are relevant, legal, ethical, and not too extravagant given available resources; Certain: stakeholders only share information about the system that they know for sure are true.	Granularity: stakeholders speak about details related to the system or the way they expect the latter to work; Rules: stakeholders speak about laws, norms, standards, their habits, their culture, or any other constraint that shaped their behaviour.	Scope: stakeholders discuss proactively about topics, even though these topics have not been suggested by the business analysts. Requirements: stakeholders discuss proactively about their requirements from the future system, even if not asked by the business analysts; Domain: stakeholders discuss proactively about the context of the system-to-be.
	Medium	Item: stakeholders speak about people, objects, concepts or other systems related with the system-to-be.	
	Light Clear: stakeholders requests are easy to understand; Priority: stakeholders share requests with a clearly defined order of priority. Concise: stakeholders share requests that are not too long, and do not take to much time to communicate them. Mandatory: stakeholders distinguish between optional and mandatory requests	Localization: stakeholders speak when and where the system will be used; Activity: stakeholders speak about the intentions of the business; Connection: stakeholders speak about relationships or links between two ore more agents of the business.	Feedback: stakeholders provide feedback about the way the project is going on, and what they believe is good or bad; Challenge: stakeholders never hesitate to challenge or criticize a decision being made by the business analysts.

Fig. 1: The Commitment Matrix

type of stakeholder, even those who are slightly commitment to the project, i.e., commitment is not important to account for when looking for clear information. However, if the analyst needs information that is certain, then she might have better chances to select strongly committed stakeholders. It is worthy to note that the survey is exploratory; our goal is to discover the impact of commitment on elicitation, and we have no a-priori hypotheses to test in the study. Further validation of the matrix is therefore required before going on the formulation of actual recommendations for the selection of stakeholders during elicitation, based on commitment profiles.

4) *Structure of the paper:* The paper explains with more details how we obtained the commitment matrix illustrated in Figure 1, and is structured as follows. In Section 2, we review related work and distinguish between the concepts of commitment and involvement. We also provide a more accurate definition of commitment, and define the commitment hierarchy. In Section 3, we describe our survey design. We present our results in Section 4, discuss them in Section 5, and finally provide a conclusion to the present work in Section 6.

II. RELATED WORKS AND DEFINITIONS

Involving stakeholders as a way to improve the chances of development success is hardly new in RE and software engineering [11], [12]. It is often acknowledged that involvement is a good way to get faster to information about the domain, and to collect more accurately information about requirements [13], [7]. It is therefore frequently recommended as a best practice during elicitation [10]. Using a case study, [14] shows the different ways in which stakeholders (and more precisely customers) may be involved, and confirms the positive influence of involvement on the acquisition of information about the requirements. In [15], it is suggested that user involvement can be improved using a model-driven requirements approach. A systematic review of empirical studies about stakeholders involvement is proposed in [7], which confirms the topic has already been extensively discussed and tested in RE. Previous works seem to agree on a definition of involvement as follows:

The act for a stakeholder to communicate directly with the analysts, to share systematic feedback and to participate actively in the different stages of the development life-cycle [16].

Involvement requires that stakeholders are correctly identified and selected. This process is another important issue during elicitation [10]. In [17], authors argue that the selection of stakeholders somehow depends on their profile; depending of the type of information the analysts look for, stakeholders with different responsibilities, skills or knowledge will be targeted. In [18], the selection of stakeholders is discussed in the context of inter-organizational environments. Among other things, the author describes a matrix to classify stakeholders based on the interest they have in the system, and on the influence they might have on this project. In [19], a systematic review of stakeholders selection techniques is proposed. As already discussed, previous stakeholders selection techniques may lead to the identification of hundreds of stakeholders, and it is hardly feasible to involve all of them during elicitation. We suggest that commitment can be used as a criterion for involving only a subset of the pool of available stakeholders.

The concept of commitment has been the center of relatively little attention in RE. The distinction between commitment and involvement is sometimes made in RE literature [7], and stakeholders' commitment is sometimes acknowledged as a factor affecting the success of RE activities; in [20] for example, it is clearly stated that the geographical dispersion of teams likely decreases commitment of the stakeholders, which may be harmful to the RE project success. The concept is however scarcely studied as a main effect influencing quality, quantity or efficiency of elicitation documentation. To the best of our knowledge, no empirical research has gone on commitment during elicitation.

Given the relatively small attention that has been paid to commitment in RE, we clarify the concept of stakeholders commitment based on more mature definitions from psychology and management sciences. We use the influential definition of organizational commitment suggested in [21] as a baseline for our definition. We use that definition because it provides a set of criteria for commitment which are easily transferable to RE projects. Starting from Mowday's definition of organizational commitment, we define stakeholders commitment to a RE project as follows: *the relative strength of a stakeholders identification with and involvement in an RE project.* As in [21], it can be characterized by at least three factors:

- *Acceptance* - A strong belief in and acceptance of the RE

TABLE I: Possible Commitment Profiles toward a RE Project

Profile	Description
Light	Stakeholders who agree with the content, purpose and values of the project, but unwilling to put much effort in the latter (<i>Acceptance</i>)
Medium	Stakeholders who agree with the content of the project, its purpose and its values, and who are willing to help under some time and/or resource constraints (<i>Acceptance</i> and <i>Effort</i>)
Strong	Stakeholders who agree with the content of the project, its purpose and its values, and who are willing to help in the project with no limitations on the time or resources (<i>Acceptance</i> , <i>Effort</i> and <i>Membership</i>)

project’s goals and values;

- *Effort* - A willingness to exert considerable effort on behalf of the RE project;
- *Membership* - A strong desire to maintain membership in the RE project.

We see these three criteria as necessary, to some extent, in order to ensure a stakeholder is committed to a RE project. In case a stakeholder respects none of these criteria, she can be assumed to be uncommitted. If she respects one or more of these criteria, the stakeholder is assumed to be somehow committed to the RE project. It is interesting to note that these three criteria build a hierarchy of commitment. For example, if there is *Effort*, then there is necessarily *Acceptance*, but not systematically *Membership*, i.e., there is a generalisation relationship between these different criteria. The commitment criteria can therefore be used to build a hierarchy of stakeholders’ commitment. In other words, it is possible to find, in the pool of available stakeholders, stakeholders who have different levels of commitment. We summarize these different possible commitment profiles in Table I.

III. EXPLORATORY STUDY - EMPIRICAL DESIGN

Our objective is to investigate how commitment affects the significance of RECs, i.e., we want to show that commitment influences the quality, quantity and efficiency of the elicitation documentation. To explore this aspect, we use a survey. Based on the results of this survey, we have been able to draw the commitment matrix, as presented in our introduction. The survey we used was composed of three main sections.

A. Procedure

1) *Assignment and Context*: The first section was intended to introduce the subject to the context of the survey and the assignments. Subjects were told the survey was intended to understand how the implication of stakeholders may help in the design of a new information system. No more details were given about the goal of the study. Subjects were put into situation with the following paragraph:

We develop a new website for internal use in your company. We need to collect information about your expectations toward that website. Our goal is to understand what you expect from such website, and what you know about the future environment of that software.

The system was then described with more details. There were two different descriptions of the website, only one of which being submitted randomly to the subject: each time a

subject opened the survey, a random number was computed that was used to select one scenario or the other, so that no subject faced the two descriptions. The objective was to provide stimuli in order to ensure the subject was clearly committed or not toward the RE project that had been presented. Although the situation was purely hypothetical, we invited the subjects to recall the last project in which they had been involved, and use it as a global context to answer our survey. Ultimately, the decision to be committed or not was left to the stakeholder, i.e., we did not force subjects to be committed or not. The two possible descriptions were as follows:

- *The website is a positive move*: “Imagine that you have been waiting for the website for ages, that it will make your job much easier, will be fun and easy to use and will ease collaboration between colleagues”;
- *The website is a negative change*: “Imagine that you are forced to use the new website, that it will change your routines and make interactions with colleagues more difficult, and that it need to be trained to use it”.

2) *Group Assignment*: The second section was intended to measure the actual commitment of the stakeholder toward the RE project. Once subjects had read the description of the RE project that had been assigned to them, we measured how committed they were toward the latter, i.e., subjects were not forced into a group and were free to be committed or not to the RE project. We classified subjects in three different groups. These groups correspond to the commitment profiles defined in Table I, namely the Light commitment group (L), the Medium commitment group (M) and the Strong commitment group (H). To allocate a subject to one of these groups, we used the three criteria of commitment presented in Section 2. For each criterion, subjects were given a binary scale equivalent to a “Yes/No” answer. The questions were as follows:

- *Acceptance*: Do you agree with the purpose of the project?
- *Effort*: Do you agree to spend time on the project?
- *Membership*: Do you want to participate on the long term to this project?

Note again that the design enables subjects to face a negative change and still be committed to the project, i.e., although we expect a positive change to lead to stronger commitment, it may happen that a stakeholder is commitment to a RE project implying a negative change, or vice-versa.

3) *Collecting Survey Data*: The third section was intended to collect data about what information stakeholders would share during the elicitation, i.e., our goal here was to actually measure RECs. The three main hypotheses we wanted to test - namely, that commitment influences quantity, quality and efficiency of the documentation - were however too coarse to be validated as such. As a consequence, we divided each REC in a series of more specific questions, easier to measure and hence to explore. We call these sub-questions *variables* in the rest of this paper. Each variable of the survey was measured using a five level Likert scale of agreement. Subjects were asked how they were feeling about a given sentence, and had to select one answer among *Strongly Disagree*, *Disagree*, *Neither Agree nor Disagree*, *Agree* or *Strongly Agree*. The set of variables that we used in our survey is reported in Table II.

The **quality of documentation REC** was studied using some of the quality criteria defined in the quality requirement framework suggested in [22]. Although the sentences shared by stakeholders during elicitation were not proper requirements, we consider the previous criteria can still be considered as ways to evaluate quality of information. The **quantity of documentation REC** was studied using some dimensions of context defined in [23]. Our objective here was simply to provide examples of topics that might be discussed during interviews, and see how subjects were behaving toward the latter. What we wanted to show is that, depending on their commitment to the RE project, stakeholders tend to discuss different topics to different extents. We do not claim such list is a way to ensure completeness of the elicitation; it simply comes as a basis to compare the groups. The **efficiency of documentation REC** is, to the best of our knowledge, a less common subject in RE, and we did not find any existing list of efficiency variables for RE. We therefore proposed some variables based on our experience. Given the exploratory nature of this study, we do not expect this decision to have a significant impact on our results.

B. Subjects

The subjects we targeted to answer previous survey are frequent users of information systems. Users represent an important proportion of the stakeholders' population, and are actors of the business whose commitment can significantly vary across a same RE project. Users therefore represents an interesting population for the present study. To make sure subjects understand the problem of sharing requirements about a system-to-be, we set three requirements on the demographic of our population: it was mandatory (i) to have at least a one-year working experience in a company, (ii) to be more than 20 years old and (iii) to be frequent user of an information system for professional use. We used Amazon Mechanical

Turk (simply MTurk hereafter) to collect data for our study. MTurk enables to access a large panel of subjects, who are sufficiently diverse to be representative of actual stakeholders' population. MTurk is based on a reward system: participants have to select some tasks that they accept to complete in exchange for a certain amount of money, determined a priori by the experimenter. We discuss the validity issues related to this approach in next section. We collected the answers of 87 subjects, all living in USA at the moment of the study: 49% of them were women, 90% were between 26 and 54 years old, and 85% had at least under-graduated. The most represented business fields were services to people (16%), retail trade (14%) and information technologies (11.5%).

C. Methodological Notes

A pre-study was performed; we submitted our survey to a dozen of stakeholders, and asked them to provide feedback about the overall readability, fluency and clarity of the survey. Based on those feedbacks, we significantly improved the survey. Answers collected during this preliminary data collection have not been included in our final data-set. The design we used in this research is a survey. As any empirical design, it may be subject to some internal and external threats. Threats to **internal validity** include the selection bias, the experimental arrangement and the confounding bias. To deal with them, we left group assignation as a responsibility of the subjects (based on commitment criteria), we randomized the position of questions within our survey, so that the combination of questions was always different for different subjects, and we paid attention to use valid scales to measure our variables. The main threat to **external validity** is the use of MTurk (see next section); we had small control on the people answering the survey, and people may not be have been sufficiently involved in answering correctly the survey. To reduce this risk, we only collected answers from Master MTurk profiles, i.e., elite groups of MTurkers who have demonstrated accuracy on specific types of HITs (i.e. survey) on the Mechanical Turk marketplace. We also made use of "attention check questions". These are questions where subjects are asked to remember some simple words and to encode the latter at the end of the survey; they enable to test attention of subjects and detect spammers. Subjects who did not answer correctly to those questions were excluded from our results.

TABLE II: REC Variables in our Survey

Quality	It is important to me to...
Clear	...explain my expectations with clear and precise words
Prioritized	...tell you which of my expectations are more/less important
Concise	...explain my expectations concisely and quickly
Feasible	...make sure what I say is relevant, legal, ethical, not extravagant
Certain	...share information I am certain of, avoid things I am not sure of
Mandatory	...make sure, during the interview, that you have understood the most important expectations I have from the system
Quantity	It is important to me to...
Items	...who will use the software, devices on which the software will run, documents which need to be used and produced by the software
Localization	...how frequently the software will be used and the location where the software will be used
Rules	...rules which apply to the software in my company, and laws and norms that the software should comply with
Connections	...the relationships between the people in my company, in order to understand my expectations from the software
Activities	...why my company needs the software, what the purpose of that software is, which problems it should help solve
Granularities	...the programming language in which the software should be made, the specifics of the databases it will manage, the components of the software, the metrics used to evaluate the quality of that software
Efficiency	It is important to me to...
Requirement	...share pro-actively information about what I expect
Domain	...share pro-actively the information I have about the environment in which the software will operate
Feedback	...give feedback about the way the project is going on
Challenge	...ask questions about choices made by the designers, or challenges decisions I do not agree with
Scope	...discuss topics other than those suggested by the business analysts

IV. ANALYSIS OF RESULTS

This section discusses the results that we obtained using our survey ¹. We perform comparison between several groups and want to see if there are significant differences between these groups. ANOVA tests can be used to achieve such conclusion. Given that our data are ordinal, we resort to the non-parametric equivalent of the ANOVA for multiple groups, namely the Kruskal-Wallis test. The null hypothesis being tested is then that none of the groups being tested statistically dominate any other one, i.e. if we can reject this hypothesis, we can conclude that there are statistical differences between at least two of the three commitment profiles. To build the commitment matrix, we put variables for which the null hypothesis cannot be rejected at the bottom of the matrix; these are the variables for which we have no indications that commitment influences their

¹Data are accessible at <http://perso.unamur.be/~cburnay/Commitment/>

value, so that they can be elicited from any type of commitment profile. Then, we look at the variables for which we can reject the null hypothesis. For these variables, we have indications that commitment influences their value, and we can then report them in the relevant layer of our matrix. Since the tests we run do not indicate the direction of the influence, we also resort to graphical representations of the survey data to complement our analysis. An important concern when measuring RECs - Quantity, Quality and Efficiency of the documentation - with several variables is the internal validity of these variables; for example, do the Clear, Concise, Certain, ... variables reliably measure the same latent Quality variable. The Cronbach's alpha is one common measure of such internal consistency. It is typically used in surveys where several Likert questions are used to build one main scale - such as in our survey - and the reliability of that scale has to be measured. It is common to interpret the Cronbach's Alpha with the following limits: a value from 0 to .50 is usually small and suggests low internal validity of the scale, a value between .50 and .70 is low but acceptable, while values above .70 are high and suggest that the scale has high internal validity. We computed an alpha of 0.7139 for the Quality scale, an alpha of 0.7322 for the Quantity scale, and an alpha of 0.8585 for the Efficiency scale. We therefore conclude our scales provide reliable ways to observe RECs.

1) *Quality Variables:* Results for the Quality REC are presented in top area of Table III. The null hypothesis can be rejected if the p-value is smaller than some significance levels, which in this study are: 1% = ***, 5% =** and 10% =*. We observe that answers from the three commitment groups only significantly differ for the Feasible and Certain variables. The tests do not enable to conclude more about the other variables that we used to measure quality. Columns a and b in Table IV show the distribution of answers across the groups (L=Low, M=Medium and H=High profile), for these two significant variables. We observe that a larger part of the strong and medium commitment profiles strongly agrees with the fact that they would pay attention to share feasible and certain information. While the difference is smaller on the agree answer, we also observe that the light commitment profile tends to disagree more frequently than the two other groups for such statements. This brings us to the conclusion that, in order to increase the chances of collecting Feasible and Certain information, business analysts should involve stakeholders with a medium commitment profile or higher.

2) *Quantity Variables:* Results for the Quantity REC are presented in the middle area of Table III. We observe that the three commitment groups significantly differ for the Items, Rules and Granularities variables. We cannot conclude anything about the other quality variables. Columns c, d and e in Table IV show the distribution of answers for the three significant variables. We observe no clear differences between the medium and strong commitment profiles, for the Items variable. This suggests that business analysts can involve any of these two profiles to reduce the risk of omissions about objects and agents who (will) interact with the system. On the contrary, we observe that strong commitment profile strongly agrees more frequently than the two other profiles when considering rules and granularities. Besides, the light and medium profiles answered more frequently that they (strongly) disagree with sharing such information. This brings us to the conclusion

TABLE III: Kruskal-Wallis Tests on the Survey Variables

Quality Variables (Freedom degree = 2)	X-Squared	P-Value	Significance
<i>Clear</i>	1.6192	0.445	-
<i>Priority</i>	2.2625	0.322	-
<i>Concise</i>	1.9774	0.372	-
<i>Feasible</i>	7.5268	0.023	**
<i>Certain</i>	5.9537	0.051	*
<i>Mandatory</i>	0.4599	0.7946	-
<i>Items</i>	4.6929	0.096	*
<i>Localization</i>	3.9085	0.142	-
<i>Rules</i>	4.8234	0.089	*
<i>Connections</i>	0.0413	0.9796	-
<i>Granularities</i>	7.6043	0.022	**
<i>Activities</i>	4.3940	0.111	-
<i>Requirements</i>	13.4056	0.001	***
<i>Domain</i>	10.6544	0.004	***
<i>Feedback</i>	3.1640	0.205	-
<i>Challenge</i>	2.7031	0.2588	-
<i>Scope</i>	9.6774	0.008	***

that analysts should involve strongly committed stakeholders if they want to reduce the risk of missing information about the constraints applying on the system or about the details related to how the system will operate.

3) *Efficiency Variables:* Results for the Efficiency REC are presented in the bottom area of Table III. The three commitment groups significantly differ for the Requirements, Domain and Scope variables. We cannot conclude more about the other efficiency variables. Columns f, g and h in Table IV show the distribution of answers across the groups, for these three significant variables. Although differences are small, we observe that the strong commitment profile seems to be more likely to (strongly) agree when being asked if they would be pro-active in sharing their requirements. This brings us to the recommendation that business analysts should involve strongly committed stakeholders if they wish to collect more efficiently information about requirements. Both medium and strong commitment groups (strongly) agree that they would share spontaneously information about the domain or the scope of the project, while the light commitment group (strongly) disagrees more frequently with these same variables. This suggests business analysts should involve stakeholders that are moderately committed (or higher) to the project, so as to increase the chances of getting efficiently information about the environment of the system, or details about its scope.

V. DISCUSSION

Our results clearly show that commitment is one factor that seems to have a potentially significant impact on the overall success of the elicitation process. However, it is interesting to note that the effect of commitment is only partial; it seems that commitment does not influence all the aspects of the elicitation challenges we identified, but only some specific concerns. The question is then to understand why a variable is impacted or not by commitment. Under such perspective, our study becomes a tool for identifying more specific research questions to be investigated in some future research. A possible interesting direction is to study the relation between the complexity of an elicitation task and the overall commitment level. For example, simple tasks like sharing clear information, speaking about connections or challenging people seem less likely to fail even when commitment profile is low. However, when the tasks require more effort - that is, when tasks are more complex -,

TABLE IV: Distribution of Quality, Quantity and Efficiency Answers by Commitment Profiles

	Certain (a)			Feasible (b)			Item (c)			Granularity (d)			Rules (e)			Requirements (f)			Domain (g)			Scope (h)		
	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H
Str. Disagree	0	0	0	0	0	0	0	0	0	5	6	2	5	0	2	0	0	0	0	0	0	0	0	0
Disagree	5	0	2	10	6	2	14	18	2	33	41	22	19	29	10	29	0	2	29	6	0	38	24	4
Neutral	19	6	4	29	0	10	19	6	8	24	12	8	19	0	12	33	18	4	19	6	8	24	0	20
Disagree	57	65	51	46	65	51	52	47	61	29	41	45	52	65	53	19	65	67	38	59	63	29	65	53
Str. Agree	19	29	43	14	29	37	14	29	29	10	0	22	5	6	22	19	18	27	14	29	29	10	12	22

commitment appears to be a more important variable. Another interesting direction is to study the link between commitment and the recency of a task. In fact, it seems that more recurring cognitive tasks such as sharing prioritized information, speaking about localizations or giving feedback are less likely to fail even if the commitment is low. In practice, these tasks are likely to occur more frequently and are likely to be more commonsense to stakeholders, so that even low commitment profiles can deal with them correctly. Overall, we believe that investigating such questions might help in the formulation of additional guidelines for the selection and involvement of stakeholders.

VI. CONCLUSIONS

In this paper, we discuss the concept of stakeholders' commitment to a RE project, why it can be a relevant criterion to select, among the large pool of stakeholders, those that should be involved in the elicitation, and how it differs from involvement. Based on the survey of 87 stakeholders, we propose the commitment matrix, which describes the relative advantages of involving stakeholders with light, medium and strong commitment profiles. We observe that stakeholders with different commitment share information which nature may vary, in terms of quality, quantity or efficiency. It suggests that, depending on the type of information a business analyst is looking for, the commitment level of a stakeholder is more or less important to account for. While the paper does not provide any elicitation methodology, it suggests ways for accounting for commitment in order to better deal with the RECs. The commitment matrix is an exploratory study, not a proper empirical validation; readers should bear in mind that it builds on a small sample, and that further validation is required before formulating recommendations for the selection of stakeholders. We believe this does not hold us back from drawing relevant conclusions about how commitment affects RECs, and about what research is necessary in the future so as to better deal with such aspect during elicitation.

REFERENCES

- [1] M. G. Christel and K. C. Kang, "Issues in requirements elicitation," *Technical Report CMU/SEI-92-TR-12 ESC-TR-92-012*, 1992.
- [2] D. Zowghi and C. Coulin, "Requirements Elicitation : A Survey of Techniques , Approaches , and Tools," in *Engineering and managing software requirements*, C. Aurum, Aybüke and Wohlin, Ed. Springer Berlin Heidelberg, 2005, pp. 19–46.
- [3] A. Sutcliffe and P. Sawyer, "Requirements elicitation: Towards the unknown unknowns," in *Proc. 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, Jul. 2013, pp. 92–104.
- [4] A. M. Davis, O. Dieste, A. M. Hickey, N. Juristo, A. Moreno, and M., "Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review," in *Proc. 14th IEEE International Conference on Requirements Engineering*, 2006, pp. 179–188.
- [5] A. M. Hickey and A. M. Davis, "A unified model of requirements elicitation," *Journal of Management Information Systems*, vol. 20, no. 4, pp. 65–84, 2004.
- [6] R. Palanisamy and J. L. Sushil, "User Involvement in Information Systems Planning Leads to Strategic Success: An Empirical Study," *Journal of Services Research*, vol. 1, no. 2, pp. 125–157, 2001.
- [7] M. Bano and D. Zowghi, "Users' involvement in requirements engineering and system success," *Proc. 3rd International Workshop on Empirical Requirements Engineering (EmpiRE)*, pp. 24–31, Jul. 2013.
- [8] A. Pouloudi, "Stakeholder analysis as a front-end to knowledge elicitation," *AI & Society*, vol. 11, no. 1-2, pp. 122–137, Mar. 1997.
- [9] R. K. Mitchell, B. R. Agle, and D. J. Wood, "Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts," *The Academy of Management Review*, vol. 22, no. 4, pp. 853–886, 1997.
- [10] M. Glinz and R. J. Wieringa, "Stakeholders in Requirements Engineering," *IEEE Software*, vol. 24, no. 2, pp. 18–20, 2007.
- [11] D. Robey and D. Farrow, "User Involvement in Information System Development: A Conflict Model and Empirical Test," *Management Science*, vol. 28, no. 1, pp. 73–85, 1982.
- [12] H. Barki and J. Hartwick, "Rethinking the Concept of User Involvement," *MIS Quarterly*, vol. 13, no. 1, pp. 53–63, 1989.
- [13] S. Kujala, M. Kauppinen, L. Lehtola, and T. Kojo, "The role of user involvement in requirements quality and project success," in *Proc. 13th IEEE International Requirements Engineering Conference*, 2005, pp. 75–84.
- [14] J. Kabbedijk, S. Brinkkemper, S. Jansen, and B. van der Veldt, "Customer Involvement in Requirements Management: Lessons from Mass Market Software Development," in *Proc. 17th IEEE International Requirements Engineering Conference*. Ieee, 2009, pp. 281–286.
- [15] J. M. Rivero, E. R. Luna, J. Grigera, and G. Rossi, "Improving user involvement through a model-driven requirements approach," *Proc. 3rd International Workshop on Model-Driven Requirements Engineering (MoDRE)*, pp. 20–29, Jul. 2013.
- [16] L. Damodaran, "User involvement in the systems design process - a practical guide for users," *Behaviour & Information Technology*, vol. 15, no. 6, pp. 363–377, 1996.
- [17] J. Coughlan, M. Lycett, and R. D. Macredie, "Communication issues in requirements elicitation: a content analysis of stakeholder experiences," *Information and Software Technology*, vol. 45, no. 8, pp. 525–537, 2003.
- [18] L. C. Ballejos and J. M. Montagna, "Method for stakeholder identification in interorganizational environments," *Requirements Engineering*, vol. 13, no. 4, pp. 281–297, Sep. 2008.
- [19] C. Pacheco and I. Garcia, "A systematic literature review of stakeholder identification methods in requirements elicitation," *Journal of Systems and Software*, vol. 85, no. 9, pp. 2171–2181, Sep. 2012.
- [20] D. Damian and D. Zowghi, "The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization," in *Proc. IEEE Joint International Conference on Requirements Engineering*, 2005, pp. 99–108.
- [21] R. T. Mowday, R. M. Steers, and L. W. Porter, "The measurement of organizational commitment," *Journal of Vocational Behavior*, vol. 14, no. 2, pp. 224–247, Apr. 1979.
- [22] A. Katasonov and M. Sakkinen, "Requirements quality control: a unifying framework," *Requirements Engineering*, vol. 11, no. 1, pp. 42–57, Oct. 2005.
- [23] C. Burnay, I. J. Jureta, and S. Faulkner, "What stakeholders will or will not say: A theoretical and empirical study of topic importance in Requirements Engineering elicitation interviews," *Information Systems*, vol. 46, pp. 61–81, Nov. 2014.

An Exploration of System Architecture on Integrating Building Management System in High-Rise Building

Zunhe LIU* and Yan LIU†

School of Software Engineering, Tongji University
Shanghai, China

Email: *logangute@gmail.com, †yanliu.sse@tongji.edu.cn,

Abstract—High-rise building is bloomed into market with a series of evolved requirements. Recently, it is of great interest to adopt the Integrating Building Management System (IBMS) in high-rise buildings. However, the current IBMS architecture solution is far from the satisfaction of performance in integrating level. In the past decade, various system integrations and collaboration technologies have been developed and deployed to application domains. In this paper, we address system sensitive problems and architecture bottlenecks by observing current architectures. We provide a generic set of solutions to support the data access flow, use of information, business systems and collaborative creation. This paper presents a typical scenario for potential system architecture in high-rise building.

Keywords—Building Management System; Integration; High-Rise Building; System Architecture;

I. INTRODUCTION

In recent years, energy costs and finite energy resource increased energy demand. At the same time, the concerns about global warming have collectively contributed to a global push for energy conservation. Modern commercial buildings have become prime targets for energy efficiency research. Improving energy efficiency in buildings has emerged as an important research area. Commercial building energy already consumes 35% of the total US electricity consumption and is estimated to rise even more[1]. This expenditure constitutes 28% energy usage in residential and 12-13% in commercial buildings. Better understanding of the building processes and designing smarter building management systems that can both maintain occupant comfort while reducing energy consumption and lead to large improvements in building operation[2].

The advent of wireless sensors has enabled buildings to be retrofit for improved monitoring of building processes and energy consumption information. This has led to several “green-building” applications such as occupancy detection[3], plug-load energy metering[4], load-disambiguation[5][6], lighting control[7], and fine-grained HVAC control[8][3]. These sensor-based applications however generate an immense amount of data. Combined with existing building control systems (such as those that run the HVAC), a significant amount of data is being generated. Unfortunately, each of these systems is closed off from each other, and thus the potential for truly interesting data analysis or control applications is lost. In fact, for many industrial systems, the data is not only inaccessible; many times it is simply thrown away.

Therefore, developing a platform to allow for applications that can span across the multiple systems that monitor and control the building environment can potentially lead to a much deeper understanding of building operations. In terms of high-rise building, as the dimension of distance and time increased, the demand and performance requirement to the building system is increased at the same time. But the lack of a structured and unifying view over the system architecture and component distribution was the main obstacle to undertaking the system design and deployment. Thus, we specifically focus on the underlying architecture and technique in order to achieve the need for performance and scalability.

In this paper, a discovery and analysis of the state-of-art in IBMS solutions is presented. At the same time, we conducted a series of onsite investigations for system architecture usage. It is envisaged that this research will give the readers insights into the critical concepts and issues for consideration in designing the systems in this area.

The remainder of this paper is organized as follows. Section 2 provides a brief introduction and situation summary about IBMS usage in our investigation building. Section 3 offers a detail discussion of IBMS and provides a generic architecture for such systems. Section 4 focuses on the challenges yet to be addressed in high-rise building system including requirements and quality attributes. Section 5 presents our preliminary 4-layers architecture of IBMS.

II. PRODUCTION REVIEW AND ONSITE INVESTIGATION

A. Production Review

We conducted a comparative study on these peer solutions from applications, protocols and architecture based on literature survey. Findings are as follows: 1) Multiple-Layered Architectures are well adopted; 2) Layering strategies may be influenced by product strengths of the provider; 3) Solutions share similar applications but with different focuses; 4) BACNet and Modbus are well supported. Current existing building management systems however greatly limit analysis and innovation potential.

These systems do not usually prioritize storing this data, and thus most of it is simply lost. These systems also tend to be independent from one another. This limits the amount of innovation that can be applied since each system only has information from its own network, and thus control algorithms tend to be simplistic. Typically, data storage and aggregation architecture are different between deployments, and account on a data server to store and later access the sensor data.

B. Onsite Investigation

We have visited a series of modern buildings to identify critical issues of the integration and collect demands from different stakeholders. And we select 4 buildings to present details of system. Brief information of these buildings has listed in Table I. We conducted the summary from 4 aspects: key features, application focus, integration level and limitations as follows. In most of these buildings, limitations and weakness are observed in the current interaction styles used for data and service integration.

C. Conclusion

Based on literature survey and onsite investigation, we have discovered that current solutions are integrated in application layer, which caused limitations and weakness. To consider challenges for high-rise buildings, IBMS performance may suffer from high data sharing demands, such as frequent data acquisition, limitation of communication capability and connection channels, high data exchanging rates across applications and so on. Additional software utilities are needed for the data integration.

III. CHALLENGES IN HIGH-RISE BUILDINGS

A. Generic Architecture

Prior to discussing improvements in IBMS in detail, it is instructive to consider BMS in terms of their popular constituent components. A generic architecture for building management systems is presented in Fig. 1. Conceptually, it can be considered consisting three key Layers.

(1) Sensor Layer: Buildings, and the electrical devices and appliances within them, are monitored by a sensor configuration that collects data and parameters.

(2) Computation Layer: Information regarding energy wastage, control and recommendations is then generated by an appropriate combination of algorithmic calculations and statistical analysis.

(3) Application Layer: This layer can be further categorized into two application sub-categories: appliance control and the provision of user feedback across a range of modalities. Implicit within this layer is a management component allowing for system testing and maintenance activities.

B. Architecture Analysis

In order to help the readers get a better understanding of the system analysis, we summarized architecture quality attributes.

TABLE I. Feature Comparison of buildings

Building ID	Key Features	Applications Focus	Integration Level
A 3 floors office	Solar energy system Fire detection system	Generate electricity Connect with Parking system by hardware	Data level
B 16 floors office	Access control system	Check work attendance	
C 32 floors 14 elevators office and classroom	Security system Elevator system	use infrared detection and dynamic cameras display real-time elevator for detection	Data level Service level
E 42 floors 6 elevator hotel	Intelligent elevator dispatching	Integration with smart video engine	Data level Service level

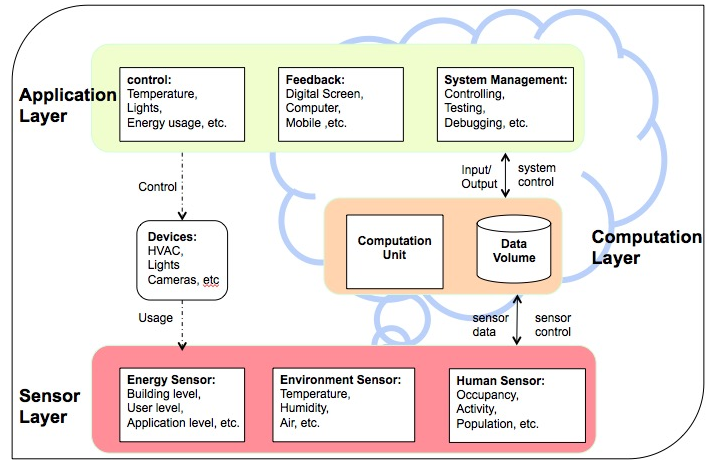


Figure 1. Generic Architecture Layers

This helps us make a profound glimpse in architecture analysis. Secondly, we discovered the potential performance bottleneck evolved if system architecture was applied to high-rise building. All the work above guide us to present an architecture design scenario specific for high-rise building system .

1) *Quality attributes*: The quality that must be ensured as part of the integration is an important criterion, as well as in high-rise building consideration. Below we identify the quality attributes that we have observed as being the most important and common in architecture integration patterns, especially which being neglected in building requirement considerations. Interoperability assures the connectivity and information interchange among systems. It concerns technology and engineering challenges related to communication, data management. Scalability requires that the integration is scalable across large numbers of systems. Thus, the integration will work correctly if more different systems are integrated.

2) *Sensitive points and bottlenecks*: In terms of business process analysis, the key problem about the system is reliance on central control in the computational layer. This leads to the constraint that changes in the application functions that affect one step might require changes in the whole process.

The system lacks data level integration. Each application in application layer requests data from the central data storage. The impact of data sharing limitation could dispread when more applications trying to request data from data center.

IV. ARCHITECTURE SPECIFICATION

After the architecture analysis in the previous section, we could discover a point that the improved architecture should be data-centric and system-friendly. The integration architecture should satisfy the quality attributes especially performance and interoperability. In this section, we will present our integration architecture scenario.

A. Architecture Description

We find out the current solution for architecture design and system application usage has a great limitation and bottleneck in high-rise building. In IBMS, it requires a high quality for

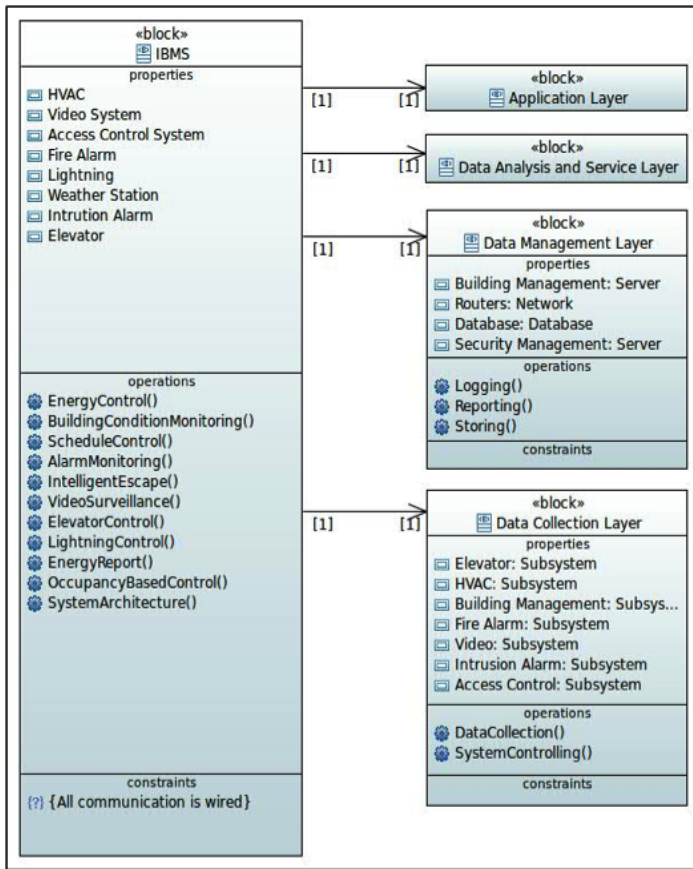


Figure 2. Four Layers of Integration System

system and architecture. Considering the huge amount of data in high-rise building and weakness in current situation, we could foresee the performance would be below the average expectations. So we propose our suggestions in this section especially for the architecture optimization. The architecture and system component is shown in Fig. 2.

- Data collection layer: it collects data from subsystems, for example elevator subsystem, HVAC subsystem and fire alarm subsystem etc.
- Data management layer: it is central storage and management point in the architecture. It takes control on how to store history data and how long the data stored in the database.
- Data analysis and service layer: it provides various service interfaces for applications. In this layer, data mining and information intelligent technology is proposed to data analysis.
- Application layer: it provides user interface for the user. And in this layer the system and program is transparent to the user. This layer integrates different systems and provides system level collaboration.

B. Architecture Improvements

We listed three main improvements respectively in different levels.

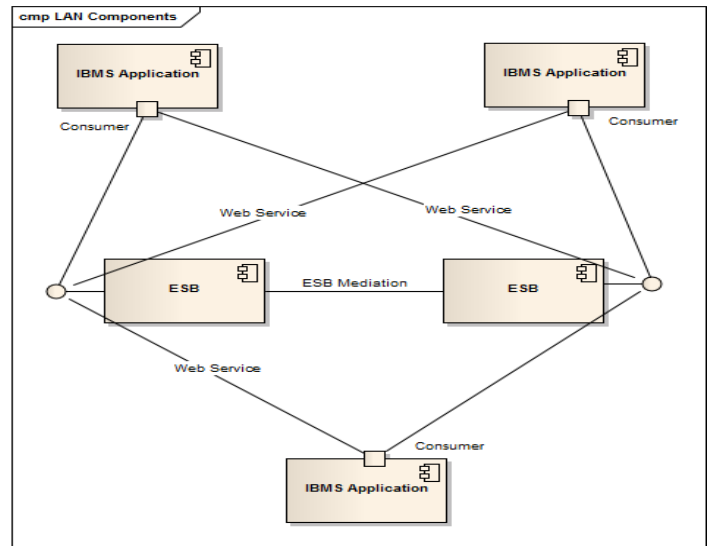


Figure 3. Middleware or ESB component for third party system integration

1. Data level: In high-rise building, the data storage and data transferring are the sensitive points in the system. The performance of them affects the whole integration system seriously. We propose a data-sharing platform. In the platform, we provide a series of data translating utilities. Meanwhile, we build data integration layer. In the integration layer, the system provides data sharing services for data reuse and data encapsulation. Integration in data level optimizes the performance in data transferring and requesting.

2. Application level: In high-rise building, there is a huge demand in application level integration, as well as application collaboration with third party system. The convention collaboration method is providing web service by using XML. To avoid this limitation, the generic architecture designs an enterprise collaboration platform. Event-driven component based architecture could be an option for the system optimization as well as applying service bus in the architecture. The platform provides service reuse and interfaces for third party system integration. The suggestion of component architecture with ESB or Middleware is shown in Fig. 3.

3. System level: This section describes the usability of system level. In the integration system, there should be some utilities and tools for the troubleshooting and debugging work in system. In case the system breaks down, the system should alert the detail information about reasons and point out the reliable solutions. Also, the system should have a surveillance tool on data transferring and correctness about data.

V. CONCLUSION AND FUTURE WORK

A. Achievements

This study identified problems and opportunities in the design, construction and operation of IBMS for high-rise buildings. We have designed questionnaires and conducted five onsite investigations to discover the real situation of IBMS. We have studied integration-centric demands, challenges, constraints, environments, contexts and potential patterns. And

based on the survey results, we have finished architectural analysis and system modeling to understand architectural issues further.

1) *Key Findings*: With the study of common applications, networking environment, typical solutions and emerging technologies in IBMS, we have discovered:

- Data conversion and Entity/Object/Device mapping in IBMS for high-rise building need enhancement in system architecture.
- high-rise building IBMS performance may suffer from high data sharing demands; additional software utilities are needed for the data integration.
- Emerging application based on IBMS solution is usually data-centric and event-driven, which should use suitable architectural styles and patterns.
- Demands on cloud platform and intelligent data processing frameworks are strong.

2) *Major Outcomes*: In this study, we have proposed a four-layered generic architecture as a basis for architectural analysis of high-rise building IBMS, identified quality attributes for IBMS and conducted quality analysis to understand IBMS architecture decision space, analyzed potential adoption of additional platform, service bus, middleware and components to improve integration quality for high-rise building, performed detailed data flow analysis to understand the architectural weakness of one specific use case, and finally explored IBMS architecture decision space from the system level.

B. Future Work

In the next phase of the research, we will make a deeper research on the data flow analysis. We will pick up some practical use cases in system, having a more specific data flow analysis in data collecting and data transferring. Meanwhile, we will explore the brand-new market thus develop some applications with user interaction data.

REFERENCES

- [1] Y. Agarwal, T. Weng, and R. K. Gupta, "The energy dashboard: improving the visibility of energy consumption at a campus-wide scale," in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 2009, pp. 55–60.
- [2] A. H. Kazmi, M. J. O'grady, D. T. Delaney, A. G. Ruzzelli, and G. M. O'hare, "A review of wireless-sensor-network-enabled building energy management systems," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 4, p. 66, 2014.
- [3] Y. Agarwal, B. Balaji, S. Dutta, R. K. Gupta, and T. Weng, "Duty-cycling buildings aggressively: The next frontier in hvac control," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*. IEEE, 2011, pp. 246–257.
- [4] X. Jiang, M. Van Ly, J. Taneja, P. Dutta, and D. Culler, "Experiences with a high-fidelity wireless building energy auditing network," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2009, pp. 113–126.
- [5] D. Jung and A. Savvides, "Estimating building consumption breakdowns using on/off state sensing and incremental sub-meter deployment," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2010, pp. 225–238.
- [6] A. Rowe, M. Berges, and R. Rajkumar, "Contactless sensing of appliance state transitions through variations in electromagnetic fields," in *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*. ACM, 2010, pp. 19–24.
- [7] D. T. Delaney, G. M. O'Hare, and A. G. Ruzzelli, "Evaluation of energy-efficiency in lighting systems using sensor networks," in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 2009, pp. 61–66.
- [8] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse, "The smart thermostat: using occupancy sensors to save energy in homes," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2010, pp. 211–224.
- [9] T. Weng, A. Nwokafor, and Y. Agarwal, "Buildingdepot 2.0: An integrated management system for building analysis and control," in *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*. ACM, 2013, pp. 1–8.
- [10] C. Voss, N. Tsikriktsis, and M. Frohlich, "Case research in operations management," *International journal of operations & production management*, vol. 22, no. 2, pp. 195–219, 2002.
- [11] H. Mintzberg, "An emerging strategy of 'direct' research," *Administrative science quarterly*, pp. 582–589, 1979.
- [12] P. Baxter and S. Jack, "Qualitative case study methodology: Study design and implementation for novice researchers," *The qualitative report*, vol. 13, no. 4, pp. 544–559, 2008.
- [13] Y. Kim, T. Schmid, Z. M. Charbiwala, and M. B. Srivastava, "Viridiscopes: design and implementation of a fine grained power monitoring system for homes," in *Proceedings of the 11th international conference on Ubiquitous computing*. ACM, 2009, pp. 245–254.
- [14] J. Lifton, M. Feldmeier, Y. Ono, C. Lewis, and J. A. Paradiso, "A platform for ubiquitous sensor deployment in occupational and domestic environments," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*. IEEE, 2007, pp. 119–127.
- [15] T. Weng, B. Balaji, S. Dutta, R. Gupta, and Y. Agarwal, "Managing plug-loads for demand response within buildings," in *Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 2011, pp. 13–18.
- [16] Y. Kim, T. Schmid, Z. M. Charbiwala, J. Friedman, and M. B. Srivastava, "Nawms: nonintrusive autonomous water monitoring system," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008, pp. 309–322.
- [17] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng, "Occupancy-driven energy management for smart building automation," in *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*. ACM, 2010, pp. 1–6.

Analyzing Exceptions in the Context of Test Data Generation Based on Symbolic Execution

Marcelo Medeiros Eler
University of Sao Paulo
Sao Paulo - SP - Brazil
E-mail: marceloeler@usp.br

Vinicius H. S. Durelli
University of Groningen
The Netherlands
E-mail: v.h.serapilha.durelli@rug.nl

Andre Takeshi Endo
Federal Technological University of Parana
Cornelio Procopio - PR - Brazil
E-mail: andreendo@utfpr.edu.br

Abstract—Testing exception scenarios is a challenging task in the context of test data generation based on symbolic execution. In such a context, test data is generated based on constraints explicitly declared in the code. However, constraints required to activate specific exceptions may not be directly declared in the code. In such a case, implicit constraints have to be inferred from exception handling mechanisms. Given that exceptions can be raised in several situations, finding constraints to generate test data to exercise all possible faulty scenarios can significantly increase the number of paths and constraints, which can cause or aggravate path explosion issues. This paper reports on an investigation that we carried out to gauge the cost (i.e., number of path constraints) of four data generation approaches aimed at covering exception dependent paths.

Keywords – Exception handling; symbolic execution; test data generation; software analysis

I. INTRODUCTION

Automatic test data generation is a notorious complex problem. Symbolic execution and constraint solving have been used as an approach to generate test data that achieve high control-flow coverage [5, 9]. During symbolic execution, program elements are represented as functions of symbolic input values [5, 9]. Each path is represented by a path constraint, which is a sequence of constraints that should be satisfied so that the underlying path can be traversed. Constraint solvers are thus used to generate concrete input values (i.e., test data) that satisfy each set of constraints.

Often, the constraints required to traverse a path are explicitly declared in the code by means of control-flow statements, such as `if` and `while`. However, some constraints are not explicitly declared in the code because they do not stem from conventional control-flow statements. Some of these constraints implicitly derive from exception handling mechanisms such as Java's `try-catch-finally` blocks. For instance, a block that declares a `NullPointerException` will be executed only when such an exception is thrown. However, in general, there are no constraint indicating in which condition such an exception will be thrown.

We classify paths that depend on an exception being thrown as *exception-dependent* paths (EDPs) [7], as oppose to *exception-free* paths (EFPs) [14]. According to an analysis performed over a sample of 100 open source projects called

SF100 [8]¹, we discovered that almost one third of the methods have at least one EDP [7]. Although the number of EDPs of a program is high, and exception handling is an important topic in software development [4], the influence of exception mechanisms to unit test data generation using symbolic execution has not been widely explored [1, 3, 5].

Taking into account the implicit constraints that stem from exception handling mechanisms can significantly increase the number of path constraints. This has the potential to exacerbate a well-known issue faced by symbolic execution approaches: path explosion [1, 5, 6], which is usually caused by complex loop structures.

The contribution of this paper is twofold. First, we discuss the characteristics and possible approaches to identify constraints that exercise EDPs and faulty scenarios. Second, we investigate how different approaches to generate test data that cover EDPs may impact the number of path constraints.

This paper is organized as follows. Section II provides background on symbolic execution and EDPs. Section III discusses possible ways to identify constraints that lead to exceptions being raised and, in turn, the execution of EDPs. Section IV describes the investigation we conducted to measure the path constraint overhead brought by approaches that generate test data tailored to execute EDPs. Section V shows related work. Finally, Section VI presents concluding remarks.

II. SYMBOLIC EXECUTION AND EXCEPTION DEPENDENT PATHS (EDPs)

Symbolic execution is a program-analysis technique that represents the elements of a program as symbolic input values [9]. Each path is represented by a path constraint, i.e., a set of constraints in a logical expression that must be satisfied in order to execute the underlying path. To generate unit test data, symbolic execution approaches resort to constraint solvers to yield solutions to the path constraints. These solutions provided by the constraint solvers are then used as test data to achieve high coverage on control-flow criteria.

Using symbolic execution and constraint solving to generate test data is appealing and straightforward. However, even though this research area has come a long way over the past

¹Details of the SF100 corpus of classes are available at <http://www.st.cs.uni-saarland.de/evosuite/SF100/>

decades, several challenges still remain [1, 5–7, 10, 16], such as path explosion and EDPs.

Path explosion is the problem of having to cope with too many paths, which can overwhelm the constraint solver and reduce the performance of the overall process [5]. The path constraints assembled during symbolic execution are usually based on individual constraints explicitly declared in the source code by means of control-flow statements. EDPs, on the other hand, are paths that can only be traversed if a specific exception is thrown. The constraints required to raise the underlying exception of an EDP, however, are not explicit in the code. Therefore, the constraints that lead to the execution of EDPs have to be abstracted from exception handling mechanisms, viz., `try-catch-finally` blocks.

Listing 1 presents an illustrative example of EDPs. The Java method `evalBMI` classifies the weight of a person as underweight, healthy weight, or overweight, given its body mass index (BMI) calculated by `calcBMI`, which might throw an `ArithmeticException`.

Listing 1: Source code of `evalBMI`.

```

1 public void evalBMI(Dialog ud, float mass, float height) {
2   winlayout.setStyle("default");
3   float bmi = 0; 01
4   try {
5     bmi = calcBMI(mass, height); 02
6     String msg = String.valueOf(bmi); 02
7     ud.print(msg); 02
8     if (bmi < 18.5) 02
9       ud.print("Underweight"); 03
10    else
11     if (bmi < 25) 04
12       ud.print("Healthy weight"); 05
13    else{
14      ud.print("Overweight"); 06
15      ud.getLayout().setColor("red"); 06
16    }
17  } 07
18  catch (ArithmeticException e) {
19    ud.print("Height must be greater than zero"); 08
20  } catch (NullPointerException e) {
21    e.printStackTrace(); 09
22  } finally {
23    ControlBoard.addBMI(bmi); 10 – 11
24  }
25 } 12

```

Notice that `evalBMI` has a `try-catch-finally` construct with two exception handlers: each `catch` block is an exception handler whose argument declares the type of exception that the handler can treat. The first handler catches an unchecked exception: `ArithmeticException`. The second handler catches another sort of unchecked exception: `NullPointerException`. The `finally` statement ensures that all instructions within its block are executed regardless of what happens in the `try` block.

In Java, unchecked exceptions inherit from either `RuntimeException` or `Error`. In general, good programming practices can avoid raising unchecked exceptions. Therefore, the compiler does not force the programmer to handle such type of exceptions, albeit it is a common practice. Notice, for example, that both `winlayout.setColor` (line 2) and `ud.print` (lines 7, 9, 12, and 14) instructions may throw a `NullPointerException`, but only the latter are within a `try` block. As oppose to unchecked exceptions, the compiler force the programmer to catch or propagate checked exceptions.

We represent the methods under test as control-flow graphs (CFGs) [11, 17]. CFGs are directed graphs in which each node usually represents a block of instructions without flow deviation (i.e., a basic block) and directed edges represent transitions (i.e., unconditional branch or jump) in the control-flow.

Figure 1 shows the CFG generated for `evalBMI`. The numbers after each instruction in Listing 1 indicate their corresponding node in the CFG. Nodes related to `try`, `catch`, and `finally` blocks are also shown. Dashed edges represent branches that are executed when some exception occurs. The exception handling mechanism of the Java language is conservative: it considers that any instruction within a `try-catch-finally` block may throw an exception. Thus, there are edges from all nodes within the `try` statement to the exception handling nodes.

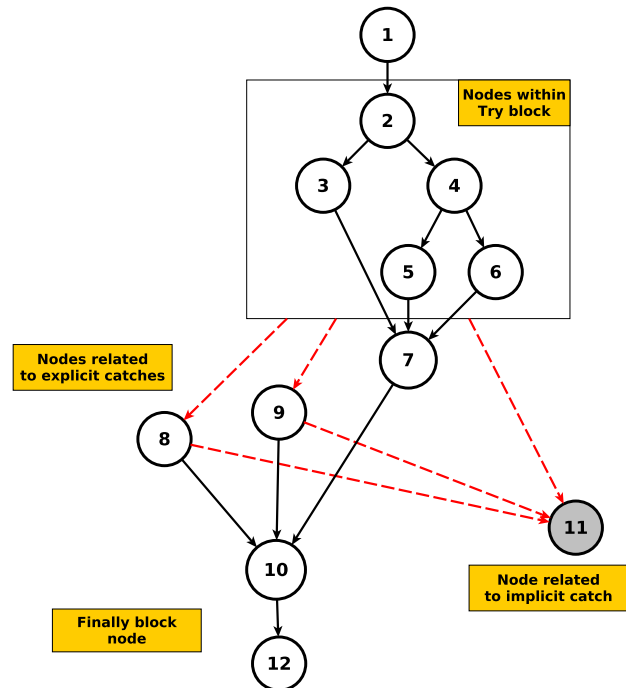


Fig. 1: CFG for `evalBMI`.

Notice that nodes 2–6 have edges to nodes 8, 9, and 11; we use only one edge from the `try` box for improving the legibility of the graph. The exception handling notation we use is adapted from the notations proposed by Sinha and Harrold [13] and Vincenzi et al. [14].

Nodes within the `try` block can reach any exception nodes (8, 9, and 11), depending on the type of the thrown exception. Node 8 catches an `ArithmeticException` and node 9 captures a `NullPointerException`. Node 11 is an implicit catch included by the compiler to capture any uncaught² exception, even those that might be thrown within `catch` statements. Node 10 represents the `finally` statement, which is reached by the exception nodes (8 and 9) and by node 7. It is important to mention that Node 11 is a copy of the `finally` block that was automatically created by the compiler to assure that the instructions under the `finally` block are executed even when an uncaught exception is thrown. Node 11 is also an exit node since it throws an exception.

During symbolic execution, we adopted a breadth-first algorithm whose goal is to generate paths that cover all branches of a CFG. Table I shows all paths of `evalBMI`, which may be either (i) EDP, when the path includes an exception edge (dashed), or (ii) EFP.

TABLE I: EDPs and EFPs of `evalBMI`.

Path ID	EDP	11	{1, 2, 4, 6, 8, 10, 12}
1	{1, 2, 11}	12	{1, 2, 4, 5, 8, 10, 12}
2	{1, 2, 4, 11}	13	{1, 2, 9, 10, 12}
3	{1, 2, 3, 11}	14	{1, 2, 3, 9, 10, 12}
4	{1, 2, 9, 11}	15	{1, 2, 4, 9, 10, 12}
5	{1, 2, 8, 11}	16	{1, 2, 4, 5, 9, 10, 12}
6	{1, 2, 4, 5, 11}	17	{1, 2, 4, 6, 9, 10, 12}
7	{1, 2, 4, 6, 11}		EFP
8	{1, 2, 8, 10, 12}	18	{1, 2, 3, 7, 10, 12}
9	{1, 2, 3, 8, 10, 12}	19	{1, 2, 4, 5, 7, 10, 12}
10	{1, 2, 4, 8, 10, 12}	20	{1, 2, 4, 6, 7, 10, 12}

Symbolic execution approaches yield path constraints based on the constraints explicitly declared along the underlying path. However, there is no constraint associated with the exception edges. Therefore, approaches that want to generate test data to exercise EDPs must implement mechanisms to derive constraints from exception handling mechanisms, otherwise they will not be covered. For instance, edge 2–9 is executed only when a `NullPointerException` is raised. In such a case, an analysis of the code would show that the constraint (`ud==null`) would result in the execution of that path.

Given that the constraints that cover EDPs are not explicit in the code, many symbolic execution techniques ignore EDPs, thereby building path constraints that take into account only explicit constraints. A clear advantage of not dealing with implicit constraints is that the number of paths to be processed is low, which can speed up test data generation. The disadvantage, however, is that many possible execution paths remain uncovered.

III. HANDLING EDPs AND UNCAUGHT EXCEPTIONS

Current symbolic execution tools and approaches do detail whether or not and how they handle EDPs. Therefore, we devised four possible approaches to generate test data that

²An uncaught exception is an exception thrown that is not captured by an exception handling mechanism. In Java, only unchecked exceptions may be uncaught.

cover EDPs and uncaught exceptions in order to increase the likelihood of covering more faulty scenarios. The impact of using these approaches is discussed in Section IV.

A. Analyzing `try-catch` Statements: Single Constraint

This approach analyzes instructions declared within `try` statements and tries to identify constraints that would throw exceptions caught by the `catch` clauses. For example, if the target exception is a `NullPointerException`, instructions that access methods and fields of an object are considered. Even though there are several instructions that could raise the target exception, only one constraint is selected for each block of instructions.

The advantage of this approach is that test data is generated considering both explicit and implicit constraints. The drawback is the increase in the number of path constraints. In addition, complex analysis techniques are required to derive constraints from exception handling mechanisms, mainly because each type of exception requires different constraints to be raised. Also, selecting only one constraint to raise the target exception may not be enough since it can be unsolvable, i.e., it is not possible to find a concrete solution to satisfy all constraints. For instance, if the constraint (`ud==null`) is selected to execute the exception edge 6–9, path {1, 2, 4, 6, 9, 10, 12} would remain uncovered since edges 2–4 and 4–6 are executed only if (`ud!=null`).

B. Analyzing `try-catch` Statements: Multiple Constraints

The analysis performed in this approach is similar to the previous approach (Subsection III-A). However, instead of selecting only one constraint for each block or node, all possible constraints are used according to the target exception. In such a case, one new path constraint is generated for each new constraint identified. For instance, two path constraints would be generated for EDP {1, 2, 4, 6, 9, 10, 12}: one considering the constraint (`ud== null`), and other considering the constraint (`ud.getLayout()==null`).

The advantage of this approach is that it explores all possible situations in which an exception can be raised within an exception handling environment. Even though many of the path constraints generated may turn out to be unsolvable, choosing several constraints increases the chances of finding test data to cover the target EDP. Nevertheless, the main drawback is that the number of path constraints yielded for each EDP may be too high, leading to path explosion.

C. Beyond `try-catch` Statements

Both aforementioned approaches are based in the fact that, in theory, programmers generally employ exception handling mechanisms in scenarios where exceptions are more likely to be thrown. According to Cabral and Marques [4], however, programmers do not catch enough unchecked exceptions making applications crash even on minor error situations. Therefore, deriving constraints only from instructions within `try-catch-finally` blocks may let several faulty scenarios uncovered due to how programmers write their code [4, 12].

In this context, we devised a thorough approach that identifies constraints that traverse EDPs by analyzing exception handling mechanisms and also constraints aimed at raising uncaught exceptions that can be raised outside the boundaries of `try-catch` statements. Yet considering a `try-catch` environment, the constraints generated are not limited to the declared exception. For each constraint identified, the underlying path constraint is replicated and the new constraint is added.

The main advantage of this approach is that it allows for yielding path constraints to generate test data that traverse EDPs and also raise uncaught exceptions. The test data generated are not limited to exercise EDPs abstracted from exception handling mechanisms. Rather, the test data generated by this approach exercises every possible faulty scenario by activating all possible exceptions.

The main drawback of this approach is the huge number of path constraints to process. This alternative can clearly aggravate the path explosion problem. Furthermore, the complexity of finding a concrete constraint to raise an exception is greater in this context since the analysis must take into account any type of exception, not only the target exceptions within `catch` statements.

Another drawback of yielding a huge number of path constraints is that many of them may be unsolvable. In such cases, resources will be spent to process constraints that will not generate any test data. One possible solution to mitigate this problem is to apply static analysis techniques to identify and eliminate unsolvable path constraints prior to sending them to the constraint solver.

D. Beyond `try-catch` Statements: An Optimized Version

We devised an optimized version of the previous approach (Subsection III-C) in which new path constraints are only created when the new constraint that activates an exception follows these rules: (i) it does not contradict any constraint in the underlying path constraint; (ii) it is not yet in the underlying path constraint; (iii) it does not raise an uncaught exception before reaching the block where the target exceptions is supposed to be thrown. The advantage of this approach is that the amount of path constraints is kept in check.

IV. STUDY OF THE OVERHEAD BROUGHT BY APPROACHES TO GENERATE TEST DATA TO FAULTY SCENARIOS

A. Study Setup and Procedure

The main goal of this study is to investigate the impact of generating unit test data to cover EDPs and uncaught exceptions, taking into account the number of path constraints for each of the four approaches presented above. Specifically, we want to find out the overhead brought on the number of path constraints.

To perform such an investigation, we selected a third party benchmark named SF100 to be the object of our investigation [8]. SF100 is made up of a collection of 100 open source Java projects that differ considerably in size, complexity,

and application domains. Altogether, these 100 Java projects contain 18,344 classes and 136,156 methods.

We used a tool called CP4SE (Constraint Profiling for Symbolic Execution) [7] to analyze the SF100 benchmark. CP4SE can symbolically execute a program under test based on its bytecode and provide the path constraints generated for each execution path. It uses a breadth-first search to find all paths considering only one loop iteration and also the aim of covering all branches. In order to tailor CP4SE for our purposes, we implemented the four test data generation approaches to cover EDPs and uncaught exceptions discussed in Section III. We adopted CP4SE as a static analysis tool, focusing on the analysis of path constraints associated to EFPs and EDPs. It is worth mentioning that the generation of test data is out of the scope of this paper.

In this study, we investigate the effects of generating path constraints related to four out of the seven most common exceptions used in the Java language according to Cabral and Marques [4]. The four exceptions we investigated are presented as follows:

- `NullPointerException`: instructions such as `obj.field` or `obj.method(...)` generate the constraint (`obj==null`), where `object` is a program element (e.g., variable or method return) whose type is an object.
- `NegativeArraySizeException`: instructions such as `array = new type[size]` generate the constraint (`size<0`), where `size` is any numeric element (e.g., variable, method return or arithmetic expression).
- `ArrayIndexOutOfBoundsException`: instructions such as `array[i]` generate the constraint (`i>=array.length`) or (`i<0`), where `array` is any array structure (e.g., variable or method return) and `i` is any numeric element.
- `ArithmeticException`: instructions such as `(x/y)` generate the constraint (`y==0`), where `y` is a numeric element (e.g., variable or arithmetic expression).

It is important to highlight that the first two approaches employed (Subsections III-A and III-B) only identify constraints for specific exception. If an instruction such as `(x/y)` is within a `try` statement caught by a `NullPointerException`, no constraint will be identified. On the other hand, all types of constraints are identified in handling mechanisms that catch generic exceptions such as `java.lang.Exception` or `AnyException`.

It is also worth mentioning that all instructions are analyzed by CP4SE after the symbolic execution of the program under test. Thus, CP4SE only identifies constraints to instructions that follow the structure defined for each exception. For example, consider a method with the instruction `(x/y)`, but the following assignment is always executed before: `y=5`. In such a case, the constraint (`y==0`) is not identified since the analyzed instruction becomes `(x/5)` after symbolic execution. The same principle holds for the other types of instructions.

TABLE II: Path constraint overhead.

Exception–Approach	No EDPs	Approach A		Approach B		Approach C		Optimized Approach C	
	# PC	# PC	Overhead	# PC	Overhead	# PC	Overhead	# PC	Overhead
NullPointerException	115,305	134,490	16.6%	149,991	30.1%	646,186	460.40%	295,701	156.5%
ArrayIndexOutOfBoundsException	115,305	125,099	8.5%	127,560	8.5%	191,414	66%	158,078	37.1%
ArithmeticException	115,305	124,150	7.7%	124,160	7.7%	126,758	9.9%	126,487	9.7%
NegativeArraySizeException	115,305	124,724	8.2%	124,768	8.2%	136,906	18.7%	130,580	13.2%
All four exceptions	115,305	134,560	16.7%	154,085	33.6%	724,233	528.1%	323,185	180.3%

B. Results

We executed CP4SE several times to generate path constraints to SF100 according to the four approaches presented in Section III and the target exceptions commented in Section IV-A. Each run of CP4SE considered a particular approach and a particular exception. Table II summarizes the results of these runs.

Rows 1 to 4 of Table II show the results for each type of exception individually, while row 5 presents the results of the four exceptions combined. The columns show the results obtained by the execution of the four approaches discussed in Section III. The first column shows how many path constraints (#PC) were identified using CP4SE when no EDP or faulty scenario is considered. This particular information is used in the rest of the table as the basis to measure the overhead brought by the implemented approaches. For each approach, we present the number of path constraints (#PC) identified and the overhead measure in percentage.

The results show that the `NullPointerException` type brings more overhead than the other three types we investigated. The overhead ranges from about 16% to 30%, when only exception handling mechanisms are analyzed. On the other hand, the overhead is increased by up to 460% when all blocks of instructions are considered. The optimizations introduced by the approach described in Subsection III-D seem to be a possible solution to this problem since the overhead dropped from 460% to 156%.

Although the number of path constraints with array elements is low, as in previous results [7], the number of instructions that uses arrays is high. As a result, the overhead brought by exceptions related to arrays is relatively high (up to 66%). The overhead regarding arithmetic exceptions is low, which is consistent with the fact that complex and nonlinear arithmetic expressions involving divisions are more frequent in specific applications according to Barr et al. [2], such as mathematical and scientific applications.

The overhead brought by the constraints identified within a `try-catch-finally` block (Subsections III-A and III-B) is significantly lower than the overhead brought by the analysis of all instructions (Subsections III-C and III-D). This means that there are several scenarios in which an uncaught exception may be thrown. This seems to agree with the analysis of Cabral and Marques [4], in which they state that, since programmers are not forced by the compiler, they do not catch unchecked exceptions properly, making applications crash even on minor error situations.

When all exceptions are considered, the overhead brought

by Approach A (Subsection III-A) is not high. As Cabral and Marques [4] remark, developers tend to catch generic exceptions. In such a case, only one exception is enough to exercise that particular path. When Approach B (Subsection III-B) is used, the overhead is a bit higher (around 33%). Considering Approach C (Subsection III-C), the overhead is extremely high (528%), which exacerbates the path explosion issue. However, when Approach D (Subsection III-D) is considered, the overhead is about 180%.

The results of our investigation show that, even considering only four types of exceptions, the overhead of thoroughly generating test data for most exception scenarios is prohibitive for many symbolic execution approaches. Practitioners and researchers must perform a carefully analysis in hopes of deciding which approach or which combination of approaches should be used in each situation.

V. RELATED WORK

Researchers have been investigating how exception handling mechanisms have been used by programmers and how these mechanisms can be tested properly. Cabral and Marques [4] carried out a quantitative study on how programmers use exception handling mechanisms. They looked at 32 projects, written in Java and .NET, and found that although the apt exceptions are thrown in most situations, programmers are not concerned with writing specialized handling code. Hindered by inflexible handling mechanisms [12], programmers fall back on writing generic exception handlers which are empty, exclusively dedicated to re-throw exceptions, or halting the method or program.

Xiaoquan et al. [15] proposed a static method to detect faults related to erroneous exception handling in Java programs. Their method combines two types of analysis: a forward flow-sensitive analysis to detect unsafe use of variables and a backward path feasibility analysis to prune false positives.

Few studies have been conducted to understand the characteristics of real-world software regarding exception handling from a symbolic execution point of view [7, 10, 16]. Xiao et al. [16] investigated path explosion in the context of dynamic symbolic execution. They analyzed the characteristics of loops in 16 open source projects written in the C# language, but their study focused on the characteristics of loops rather than their overall presence and relation with exceptions.

In a previous paper [7], we studied the distribution of path explosion, constraint complexity, dependency, and EDPs over the SF100 benchmark [8]. Regarding path explosion, the impact caused by loops and nested loops was investigated. Concerning

EDPs, we found out that 36% of the analyzed methods of SF100 had at least an EDP, but the impact on the number of path constraints generated has not been analyzed.

The main difference between our study and the related research is the investigation of how generating test data to cover EDPs can impact path explosion according to three different approaches. To the best of our knowledge, no other studies on this subject has been carried out.

VI. CONCLUDING REMARKS

Although symbolic execution has been extensively investigated as a promising approach for test data generation, little research has taken into account the generation of test data that cover exception-related paths. Despite the fact that many paths in a program are *exception-dependent* (i.e., EDPs), most approaches have focused on *exception-free* paths (i.e., EFPs). In this paper, we investigated this topic by looking at the increase in the number of path constraints resulted from four different test data generation approaches that cover exception scenarios.

The results would seem to suggest that the overhead caused by common exceptions, as `NullPointerException` and `ArrayIndexOutOfBoundsException`, is high, while the overhead caused by the other two exceptions is relatively low. When the four investigated exceptions are considered together, the overhead may be manageable if constraints are derived only from `try-catch-finally` statements (around 33%). However, it may be impracticable if constraints are derived from all instructions of the program under test (around 180%).

In conclusion, we believe that practitioners and researchers that want to generate test data tailored to cover exception-based scenarios should evaluate the trade-offs of using a thorough approach: generating test data for most likely exception-based scenarios results in a considerable overhead; on the other hand, focusing only on instructions declared within exception handling mechanisms or eschewing certain exceptions (e.g., `NullPointerException`) may leave many faulty scenarios uncovered.

ACKNOWLEDGMENTS

The authors would like to thank the financial support provided by CAPES (BEX 1714/14-7), FAPESP (2014/08713-9), and CNPq (445958/2014-6).

REFERENCES

[1] S. Anand, E. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. McMinn. An orchestrated survey on automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[2] E. T. Barr, T. Vo, V. Le, and Z. Su. Automatic detection of floating-point exceptions. In *Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 549–560, New York, NY, USA, 2013.

[3] E. Bounimova, P. Godefroid, and D. Molnar. Billions and billions of constraints: Whitebox fuzz testing in production. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 122–131, Piscataway, NJ, USA, 2013. IEEE Press.

[4] B. Cabral and P. Marques. Exception handling: A field study in java and .net. In *Proceedings of the 21st European Conference on Object-Oriented Programming*, pages 151–175, Berlin, Heidelberg, 2007. Springer-Verlag.

[5] C. Cadar and K. Sen. Symbolic execution for software testing: three decades later. *Communications of the ACM*, 56(2):82–90, 2013.

[6] C. Cadar, P. Godefroid, S. Khurshid, C. S. Păsăreanu, K. Sen, N. Tillmann, and W. Visser. Symbolic execution for software testing in practice: Preliminary assessment. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1066–1071. ACM, 2011.

[7] M. M. Eler, A. T. Endo, and V. H. S. Durelli. Quantifying the Characteristics of Java Programs that May Influence Symbolic Execution from a Test Data Generation Perspective. In *The 38th Annual Int. Computers, Software & Applications Conference*, pages 181–190, 2014.

[8] G. Fraser and A. Arcuri. Sound Empirical Evidence in Software Testing. In *Proc. of the 2012 Int. Conf. on Software Engineering*, pages 178–188, 2012.

[9] J. C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, July 1976.

[10] X. Qu and B. Robinson. A Case Study of Concolic Testing Tools and their Limitations. In *International Symposium on Empirical Software Engineering and Measurement*, pages 117–126, 2011.

[11] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, 11(4):367–375, 1985.

[12] M. P. Robillard and G. C. Murphy. Designing Robust Java Programs with Exceptions. *ACM SIGSOFT Software Engineering Notes*, 25(6):2–10, 2000.

[13] S. Sinha and M. Harrold. Criteria for Testing Exception-Handling Constructs in Java Programs. In *Proc. of the Int. Conf. on Sw Maintenance*, pages 265–274, 1999.

[14] A. M. R. Vincenzi, M. E. Delamaro, J. C. Maldonado, and W. E. Wong. Establishing structural testing criteria for java bytecode. *Software Practice & Experience*, 36(14):1513–1541, 2006.

[15] X. Wu, Z. Xu, and J. Wei. Static Detection of Bugs Caused by Incorrect Exception Handling in Java Programs. In *11th International Conference on Quality Software (QSIC)*, pages 61–66, 2011.

[16] X. Xiao, S. Li, T. Xie, and N. Tillmann. Characteristic studies of loop problems for structural test generation via symbolic execution. In *Proc. 28th IEEE/ACM Int. Conf. on Automated Software Engineering*, November 2013.

[17] H. Zhu, P. A. V. Hall, and J. H. R. May. Software Unit Test Coverage and Adequacy. *ACM Computing Surveys*, 29(4):366–427, 1997.

Automatically Evaluating the Efficiency of Search-Based Test Data Generation for Relational Database Schemas

Cody Kinnerer [★]

Gregory M. Kapfhammer [★]

Chris Wright [☆]

Phil McMinn [☆]

[★] Allegheny College

[☆] University of Sheffield

Abstract

The characterization of an algorithm's worst-case time complexity is useful because it succinctly captures how its runtime will grow as the input size becomes arbitrarily large. However, for certain algorithms—such as those performing search-based test data generation—a theoretical analysis to determine worst-case time complexity is difficult to generalize and thus not often reported in the literature. This paper introduces a framework that empirically determines an algorithm's worst-case time complexity by doubling the size of the input and observing the change in runtime. Since the relational database is a centerpiece of modern software and the database's schema is frequently untested, we apply the doubling technique to the domain of data generation for relational database schemas, a field where worst-case time complexities are often unknown. In addition to demonstrating the feasibility of suggesting the worst-case runtimes of the chosen algorithms and configurations, the results of our study reveal performance trade-offs in testing strategies for relational database schemas.

1 Introduction

Many disciplines, such as science, finance, and medicine, rely on relational databases to maintain large amounts of critical information [1]. The relational database schema defines the structure of a database and protects the integrity of the data. This makes testing the database schema necessary to avoid the corruption of data. Search-based algorithms, that use a fitness function to offer guidance to a desirable solution, have been applied to this challenging problem [2]. Although data generation for relational schemas may also be handled, albeit less effectively, with random generation techniques [3], the use of search-based approaches ensures that data creation methods can actively seek out test inputs that best fulfill testing goals [4].

Despite the effectiveness of search-based data generation methods, there is, to the best of our knowledge, little prior research that fully studies their efficiency and characterizes their worst-case time complexity. In part, we attribute this dearth of past work to the fact that these systems are complex, thus making a generalizable theoretical analysis hard.

In response to the lack of insight into the performance of search-based methods, this paper presents a fully automated performance evaluation framework that employs

doubling experiments to suggest worst-case time complexities and conditional inference trees to identify efficiency trends. Applying this framework to the automated performance evaluation of search-based test data generation for database schemas, the results reveal trade-offs in efficiency with respect to the chosen testing goals, the structure of the relational schema, and the data generation strategy.

Since the presented approach is fully automated, it enabled a comprehensive study suggesting the worst-case time complexity of all the relevant data generator configurations. Although this paper focuses on automatically evaluating the efficiency of search-based test data generation for the database schema, the presented technique can be applied to a wide range of other methods using heuristic search. In summary, this paper's important contributions include:

1. A performance evaluation framework that automatically conducts and analyzes the results from doubling experiments with search-based methods.
2. Empirically derived suggestions for the worst-case time complexity of search-based test data generators.
3. With a systematic focus on a wide variety of configurations, an empirical study revealing trade-offs in search-based test data generation for relational schemas.

2 Background and Related Work

Testing Database Schemas. The relational database, a cornerstone of modern software, is protected by a schema that defines integrity constraints ensuring the coherence of data. These constraints defend the schema from manipulations that could violate requirements such as “user names must be unique” or “the host name cannot be missing or unknown”. Prior work in this area proposed coverage criteria, derived from logic coverage criteria, that establish different levels of testing for the formulation of integrity constraints in a database schema [3]. These range from simple criteria that mandate the testing of successful and unsuccessful INSERT statements into tables to more advanced criteria that test the formulation of complex integrity constraints such as multi-column PRIMARY KEYS and arbitrary CHECK constraints. This family of criteria has been organized into a subsumption hierarchy, with criteria such as *Clause-Based Active Integrity Constraint Coverage* (ClauseAICC) emerging as a stringent testing strategy. Space constraints limit further commentary on testing methods for database schemas; prior work [3] provides additional details.

Ratio $f(2n)/f(n)$	Worst-Case Conclusion
1	constant or logarithmic
2	linear or linearithmic
4	quadratic
8	cubic

Table 1: Conclusions for worst-case time complexity.

Search-Based Test Data Generation. When testing a schema’s integrity constraints for correctness, it is often necessary to provide input to the database and then observe and evaluate its execution [2]. Since the database’s behavior is dependant on the input from INSERTs, the input space must be sufficiently explored to ensure thorough testing. Due to the fact that it is challenging to manually create input that supports high-quality testing, test data generation is used to automatically produce it according to a criterion, like ClauseAICC. A search-based test data generator is one that explores that input space using, among other components, a fitness function that rates the data’s quality, thus allowing it to improve by repeatedly seeking better inputs [4].

Worst-Case Time Complexity. A useful understanding of an algorithm’s efficiency, the worst-case time complexity gives an upper bound on how an increase in the size of the input, denoted n , increases the execution time of the algorithm, $f(n)$. This relationship is often expressed in the “big-Oh” notation, where $f(n)$ is $O(g(n))$ means that the time increases by no more than on order of $g(n)$. Since the worst-case complexity of an algorithm is evident when n is large [5], one approach for determining the big-Oh complexity of an algorithm is to conduct a doubling experiment with increasingly bigger input sizes. By measuring the time needed to run the algorithm on an input of size n and the time needed to run with input of size $2n$, the algorithm’s order of growth can be empirically determined [5, 6].

The goal of a doubling experiment is to draw a conclusion regarding the efficiency of the algorithm from the ratio $f(2n)/f(n)$ that represents the factor of change in runtime from input sizes n to $2n$. For instance, a ratio of 2 would indicate that doubling the input size resulted in the runtime’s doubling, thus leading to the conclusion that the algorithm under study is $O(n)$ or $O(n \log n)$. Table 1 shows some common time complexities and their corresponding ratios.

Related Work. Goldsmith et al. [7] developed a tool, called *Trend-Prof*, that empirically evaluates the computational complexity of a program by using code instrumentation to count the number of times each block of code is executed and then grouping these blocks by their behavior. *Trend-Prof* takes in a collection of workloads, user-specified features of the workloads, and the program to be studied. While this technique results in a more detailed analysis than the one presented in this paper, Goldsmith et al. did not address the issue of generating the workloads necessary to achieve a meaningful result, which this paper’s technique can handle automatically. Our paper is also con-

trasted with this prior work because it describes experiments in a domain, search-based test data generation, where the method’s worst-case time complexity is not always known.

Zhao et al. presented an empirical study of the performance of search-based test data generation for extended finite state machine (EFSM) models [8]. Although this paper focused on efficiency and made preliminary observations about the relationship between performance and the characteristics of an EFSM model, it did not, like our paper, use doubling experiments to suggest worst-case time complexities. Lakhotia et al. also reported on an experimental analysis of the efficiency and effectiveness of search-based test data generation for C programs [9]. While our paper looks at generator performance in a holistic manner, this prior work considered the number of fitness evaluations during data generation. Similar to our use of doublers that systematically increase the size of a schema, Mehrmand and Feldt empirically studied, with a focus on code coverage, search-based data generation as program size increased [10].

The empirical work presented in this paper is complemented by theoretical runtime analyses in prior research. For instance, Arcuri presented the first runtime analysis of a search-based test data generator called the alternating variable method (AVM) [11], which is also studied in this paper. Arcuri proved the worst-case time complexity of AVM when it generates data for a simple program called “triangle classification”. More recently, Kempka et al. extended the work of Arcuri with a theoretical and empirical runtime analysis revealing that the use of certain local search techniques with AVM yields better performance than AVM alone [12]. While our paper’s automated framework can easily be applied to new schemas—and even to other types of search-based test data generators—the results in these two aforementioned papers are more difficult to generalize.

3 Automated Doubling Experiments

Overview. The presented technique for performing automatic doubling experiments consists of two key components. The first is a method for systematically doubling the initially input relational schema, and the second is a rule for determining when a valid conclusion can be drawn from the experiment, thus allowing the doubling process to stop.

Doubling Schemas. Determining worst-case complexity by a doubling experiment requires that the size of the input be doubled. A relational database schema is a complex artifact with many features and interrelationships. This makes doubling rule implementation a non-trivial task.

A relational database schema contains tables and columns, and constraints that restrict the values allowed into these entities. Since the runtime of a schema testing technique may be affected by the number of any of these, it is desirable to have a strategy for doubling each one. Doubling the number of tables or columns in a schema is relatively easy. It is possible to double the number of tables

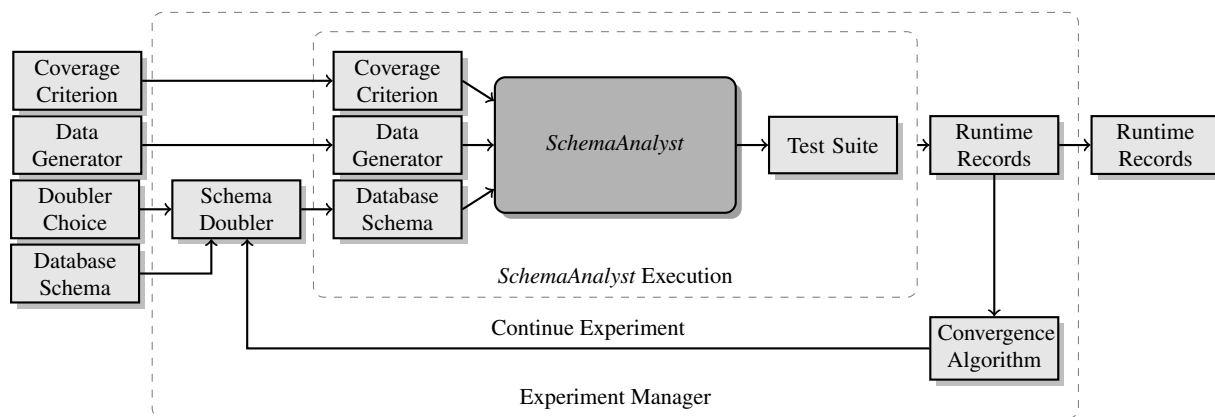


Figure 1: Technique for conducting automatic doubling experiments.

in a schema by following this rule: for every table present in the schema, create a new empty table. It is important that the new tables be empty to avoid changing more than one doubling parameter at once—if the new tables contained columns, for instance, then the number of tables and columns in the schema both would be increased, thus interfering with assessing table doubling’s impact on performance. Additionally, doubling the number of columns can be accomplished by, for every table in the schema, and for every column, adding a new column to that table.

Doubling integrity constraints is more challenging. The FOREIGN KEY constraint, for instance, denotes a relationship between two tables, thus making it difficult to double without introducing extraneous database entities or cyclic dependencies. Since a CHECK constraint can express arbitrary conditions, it is also challenging to double if the meaning of each constraint must be considered to ensure satisfiability. Since a table can only contain one PRIMARY KEY, if a schema contains five tables, then at most it can have five PRIMARY KEY constraints, as adding more keys would require creating more tables, which should be avoided.

Because of these issues, and others like them, we focus our attention on constraints that can be doubled as follows: for every table and for every constraint, duplicate that constraint and re-add it to the table. Constraints such as NOT NULL, UNIQUE, and CHECK are amenable to doubling in this fashion. It is worth noting that, since they amount to a restatement of existing constraints, entities doubled this way would not have an impact on what data the schema would allow or disallow into a database. However, since the goal is to evaluate performance, the timing results should not be adversely affected as long as the test data generation technique must still process and consider these additional constraints.

Automatic Experimentation. To determine worst-case complexity, an input of size n is doubled until the ratio $f(2n)/f(n)$ converges to a stable value. To account for random error, every time n is doubled, $f(n)$ is computed ten times and the median time is used for calculating the ratios;

we chose the median to minimize the effect of outliers. If the mean is used instead, then a single abnormally long run could have an outsized impact on the result. Figure 1 shows the overall structure of the experimentation framework.

Convergence checking is necessary because of the fact that worst-case time is only evident for large values of n . If too few doubles are tried, then the experiment may terminate before n reaches a value where the true worst-case time complexity is apparent. At the same time, for inefficient algorithms, each additional doubling run incurs a substantial time overhead. For the sake of efficiency, the doubling experiment should terminate as quickly as is possible.

To test for convergence, for every time t , where t denotes the number of times the input has been doubled, we record the doubling ratio $r_t = \frac{f(2^t n)}{f(2^{t-1} n)}$. The current ratio r_c is compared to a previous ratio r_p where p is determined by a *lookback* value, such that $p = c - \text{lookback}$. The result of the comparison is a *difference* value, given by $\text{difference} = |r_c - r_p|$. This is then compared to a *tolerance* value, and the experiment is judged to have converged when $\text{difference} < \text{tolerance}$. The *lookback* and *tolerance* values are both configured before the experiment is run.

Another consequence of worst-case time only being apparent for large n is that a very small initial n may appear to converge to 1, which would indicate constant time complexity. To prevent the experiment from incorrectly terminating given a small starting n , our method requires that a program under study display a ratio of 1 for a *minimum* number of times before judging that the ratio does in fact converge to 1. That is, if $r_c = 1$, $t > \text{minimum}$ must be true, in addition to the tolerance test, before the experiment is declared convergent. The *minimum* parameter is also configured before an experiment. Because a doubling ratio of 1 signifies constant or logarithmic time complexity, requiring these doubles does not significantly increase the experimentation time needed, all the while providing further assurance that a small ratio is not due to an insufficiently small n .

Schema	Tables	Columns	Constraints
BioSQL	28	129	186
Cloc	2	10	0
iTrust	42	309	134
JWhoisServer	6	49	50
NistWeather	2	9	13
NistXTS748	1	3	3
NistXTS749	1	3	3
RiskIt	13	57	36
UnixUsage	8	32	24

Table 2: Database schemas used in the experiments.

4 Empirical Analysis

Experimental Design. To gain a full picture of the performance trade-offs, we conducted an experiment for every configuration of the parameter space (i.e., schema, coverage criterion, data generator, and doubling technique). Table 2 shows that the experiments focused on nine database schemas containing between 1 and 42 distinct tables, 3 to 309 columns, and up to 186 constraints. Including all of the test adequacy criteria proposed by McMinn et al. [3], the experiments study “weak” criteria (i.e., APC, NCC, ICC, and UCC), “moderately strong” ones (i.e., ANCC, AICC, and AUCC), and “strong” criteria (i.e., CondAICC and ClauseAICC). More details about each criterion, including its formal definition and relationship to the other criteria, are available in prior work [3]. We used all six test data generators provided by the *SchemaAnalyst* tool for automated test data generation [2], with four techniques employing a variant of random search and two based on Korel’s alternating variable method. After a restart of the search, each data generator could start with either default or random values.

In our study, we set *tolerance* to 0.40 and *lookback* to 4. These values were chosen by performing doubling experiments on various algorithms, with known worst-case time complexities, and observing that the ratio converged to the correct value with this configuration. After observing that *SchemaAnalyst* stopped displaying constant behavior after around five doubles, we set *minimum* to be four times this number. Preliminary studies showed that, while experiments for “fast” configurations could be completed in less than an hour, “slower” configurations required days. Since there are over two thousand possible configurations, the study needed a substantial amount of computational resources. As a solution, we ran the experiments on a high-performance computing (HPC) cluster containing 195 worker nodes of various hardware configurations, ranging from 12 to 16 CPU cores and 24 to 256 GB of memory, and using a 64-bit GNU/Linux operating system.

Results. Our experiments reveal that, when doubling UNIQUES, NOT NULLs, and CHECKs, *SchemaAnalyst* displays linear or linearithmic worst-case time complexity. Out of the 699 experiments performed to double these schema structures, 72% converged to linear or linearithmic. Another 8% failed to converge, and of these experiments, 80%

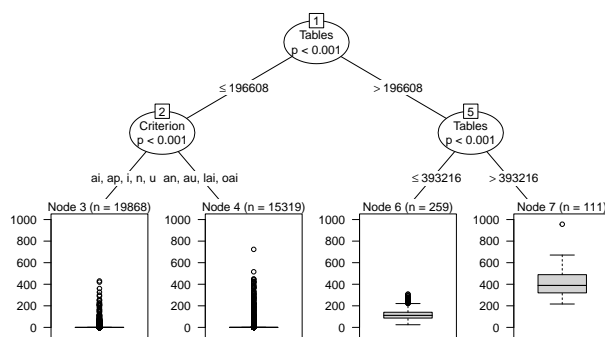


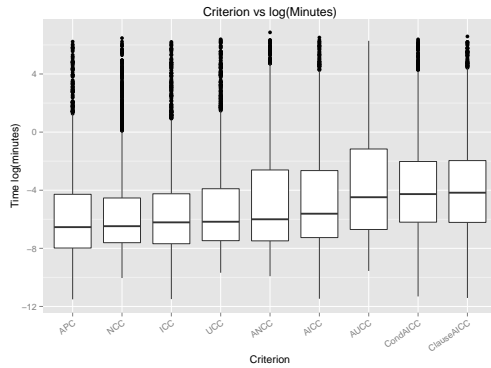
Figure 2: Tree model using all variables to predict runtime in minutes, demonstrating the important of the table count. Due to space constraints, criterion names are abbreviated.

failed because of memory limitations, 13% exceeded the maximum time limit, and 8% failed for reasons that could not be determined. The doubling ratios among these experiments were primarily linear or linearithmic at the time they were terminated, however there were 14 that were quadratic and 3 that were cubic. The experiments that failed to converge were primarily generating test data for complex schemas, such as iTrust and BioSQL, and the most stringent adequacy criteria, such as ANCC and AUCC. The remaining 20% of the 699 experiments converged on constant or logarithmic. Since there did not seem to be a pattern in which configurations converged this way compared to linear or linearithmic, it is likely that they terminated before the true worst-case time complexity was apparent.

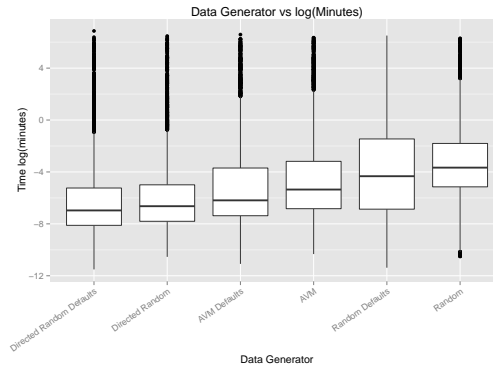
When doubling the tables and columns in the schemas, the results were less conclusive. Doubling the number of tables in the schema caused the runtime of *SchemaAnalyst* to increase much faster than it did for the other integrity constraints. As a result, 56% of the 467 experiments doubling this schema feature were terminated before convergence because they exceeded the time limit. Of the experiments that converged, 72 converged to quadratic and 10 converged to cubic. Of the experiments that terminated before they converged, the doubling ratios for 205 indicated quadratic, 18 suggested cubic, and 37 were worse than cubic.

Experiments on the number of columns were also inconclusive. We noted that 208 of the converged experiments showed linear or linearithmic time complexity, while 28 converged to quadratic and 2 cubic. Another 203 experiments failed to converge; however, unlike the experiments that doubled the number of tables, the experiments for doubling the number of columns most frequently failed by running out of memory rather than exceeding the time limit. The experiments that did not converge included 106 ratios indicating quadratic behavior, 73 cubic, and 3 worse.

To gain a more nuanced understanding of the results, our tool constructed a conditional inference tree model using



(a) Coverage criterion versus runtime in minutes.



(b) Data generator versus runtime in minutes.

Figure 3: Box and whisker plots for criterion and data generator.

the *ctree* package in the R language. These trees use the values of predictor variables (e.g., the adequacy criterion) to model the value of a response variable (e.g., *SchemaAnalyst's* runtime); *ctree* accomplishes this by repeatedly splitting the data according to what predictor variable has the most influence on the response variable. Each tree node represents a choice of predictor variable, and the level of the node indicates its importance to the prediction, with higher nodes being more important to predicting generation time.

Using predictor variables for the number of tables, columns, UNIQUEs, NOT NULLs, and CHECKS; and the chosen criterion and data generator, *ctree* produced the tree model in Figure 2. In addition to confirming that the number of tables has the greatest impact on runtime, the tree also reveals that, when the number of tables in the schema is small, the choice of coverage criterion is most significant. While the number of tables had a large impact when over 197,000, in practice schemas are unlikely to be this large. Another invocation of *ctree*, excluding tables from the list of predictors, provided insight into the behavior of *SchemaAnalyst* for more practical table sizes. In this tree, not shown due to space constraints, the coverage criterion emerged as the most important predictor for runtime, followed by the choice of data generator, and then the number of columns.

While the trees provide insight into the relative impact of each predictor, the box and whisker plots shown in the leaves of the trees do not furnish a detailed view of the choices within each predictor. To gain a finer-grained understanding, we created box and whisker plots of our own: Figure 3a shows the influence of coverage criterion on runtime, while Figure 3b shows the effect of data generator on runtime. Figure 3a shows that the strongest coverage criteria in the subsumption hierarchy (i.e., AUCC, ClauseAICC, and CondaAICC) cause runtime to increase the most, followed by ANCC and AICC, and then the remaining criteria (i.e., APC through UCC). We anticipate that the stronger criteria always lead to higher time overheads because they force *SchemaAnalyst* to generate more tests. Also, criteria at the same level in the hierarchy engender similar runtimes.

Figure 3b reveals that, by a substantial margin, the Random and Random Defaults generators took the most time to generate data. This counterintuitive result suggests that less effective data generators actually take longer to create data than those that are known to be more effective [2]. A less pronounced difference between the remaining generators can be observed, with the use of default values consistently being faster than the use of random values at restart.

While the box and whisker plots show how choices between coverage criteria and data generators affect runtime, the question remains if these differences are statistically and practically significant. To answer this question, we employ the Wilcoxon rank-sum test and the \hat{A}_{12} effect size [3].

The Wilcoxon rank-sum test is a non-parametric test for hypothesis testing. If the result of the test is greater than the significance level (0.05 is frequently used), then the configurations are indistinguishable. If, however, the result is less than the chosen level, then they are different. The \hat{A}_{12} test is similar, but for drawing conclusions about the practical difference between two collections of data. A result of $\hat{A}_{12} = 0.50$ means that any difference is not practically significant, while $\hat{A}_{12} > 0.56$ or < 0.44 signifies a small difference, $\hat{A}_{12} > 0.64$ or < 0.36 denotes a medium difference, and $\hat{A}_{12} > 0.71$ or < 0.29 indicates a large difference.

Table 3 shows the statistical tests calculated for every pair of coverage criteria. The Wilcoxon rank-sum test reveals that changing the criterion results in statistically significant differences in runtimes, with the exception of changing between the four criteria at the top of the subsumption hierarchy and the two criteria at the bottom. The \hat{A}_{12} results generally show a small to medium practical effect size when switching between criteria at the high or low end of the hierarchy, and small or no effect when switching between criteria at the same level of the hierarchy.

The statistical tests were calculated for all pairs of data generators, but the resulting table was omitted due to space constraints. All comparisons of data generators were statistically significant according to the Wilcoxon rank-sum test. The \hat{A}_{12} values show that all choices of data gener-

	APC	ANCC	CondAICC	NCC	AUCC	AICC	ClauseAICC	ICC	UCC
APC	NA	0.425	0.337	0.484	0.334	0.413	0.329	0.481	0.449
ANCC	2.20E-16	NA	0.407	0.561	0.405	0.484	0.399	0.554	0.526
CondAICC	2.20E-16	2.20E-16	NA	0.671	0.503	0.581	0.492	0.656	0.634
NCC	1.20E-02	2.20E-16	2.20E-16	NA	0.335	0.417	0.322	0.491	0.461
AUCC	2.20E-16	2.20E-16	6.92E-01	2.20E-16	NA	0.577	0.490	0.651	0.628
AICC	2.20E-16	1.70E-02	2.20E-16	2.20E-16	2.20E-16	NA	0.412	0.571	0.547
ClauseAICC	2.20E-16	2.20E-16	2.72E-01	2.20E-16	1.40E-01	2.20E-16	NA	0.662	0.641
ICC	4.00E-03	2.20E-16	2.20E-16	1.83E-01	2.20E-16	2.20E-16	2.20E-16	NA	0.472
UCC	9.30E-16	3.83E-05	2.20E-16	7.36E-10	2.20E-16	5.73E-13	2.20E-16	9.29E-06	NA
Rank-Sum:		significant	insignificant		\hat{A}_{12} :	none	small	medium	large

Table 3: For each pair of coverage criteria, lower left shows Wilcoxon Rank-Sum Test, upper right shows \hat{A}_{12} .

ator have at least a small practical impact, with the exception of choosing between random and random defaults, and directed random and directed random defaults. Changing between these data generators results in a large to medium effect size, and comparing either of the AVM-based generators to the other primarily resulted in a small difference.

Threats to Validity. Our technique for doubling the number of constraints in the schema is simply to duplicate the existing constraints. It is possible that *SchemaAnalyst* does less work processing these redundant constraints than it would given non-repeated ones. However, doubling the constraints in this way is easy to implement and, as the results show, good at revealing performance trade-offs. Additionally, since worst-case time is only apparent for large n , it is possible that the experiments terminated too quickly. While we attempted to configure the parameters of our tool using algorithms with known worst-case complexities and conducting preliminary experiments with various settings and under manual supervision, it is possible that our configuration was not optimized for use on the HPC cluster.

5 Conclusions and Future Work

This paper presented an automated method for empirically suggesting the worst-case time complexity of search-based test data generation methods. Focusing on the domain of relational database schemas, our approach repeatedly doubles the size of the input schema and observes the commensurate change in runtime. Although some results are inconclusive, we find that, in many cases, data generation is linear or linearithmic and, in others, it is quadratic, cubic, or worse. Our automated method also revealed that, for all of the test adequacy criteria in the subsumption hierarchy presented by McMinn et al. [3], stronger criteria always necessitate more time for test data generation.

Since this paper’s technique did not consider the doubling of constraints like FOREIGN KEYS, future work will focus on creating doublers for these unstudied constraints. Additionally, the current doubling mechanism avoids introducing semantically invalid constraints by restating existing constraints; in future work we plan to implement and evaluate more realistic ways to double relational schemas. Because certain experiments timed out before converging, we

also want to re-run these configurations with longer time limits and more memory. Finally, we will investigate how automated parameter tuning, instead of manual tuning before experimentation in a new execution environment, can support choosing the convergence condition. Ultimately, the combination of the presented framework with the completed future work will yield an effective way to empirically understand the worst-case case time complexity of search-based test data generation for relational database schemas.

References

- [1] G. M. Kapfhammer, “A comprehensive framework for testing database-centric applications,” Ph.D. dissertation, University of Pittsburgh, 2007.
- [2] G. M. Kapfhammer, P. McMinn, and C. J. Wright, “Search-based testing of relational schema integrity constraints across multiple database management systems,” in *6th ICST*, 2013.
- [3] P. McMinn, C. J. Wright, and G. M. Kapfhammer, “An analysis of the effectiveness of different coverage criteria for testing relational database schema integrity constraints,” Department of Computer Science, University of Sheffield, Tech. Rep., 2015.
- [4] P. McMinn, “Search-based software test data generation: A survey,” *Soft. Test., Verif. and Reliab.*, vol. 14, no. 2, pp. 105–156, 2004.
- [5] C. C. McGeoch, *A Guide to Experimental Algorithmics*. Cambridge University Press, 2012.
- [6] R. Sedgewick and M. Schidlowsky, *Algorithms in Java: Fundamentals, Data Structures, Sorting, Searching*, 3rd ed. Addison-Wesley Longman Publishing Co., Inc., 1998.
- [7] S. F. Goldsmith, A. S. Aiken, and D. S. Wilkerson, “Measuring empirical computational complexity,” in *6th ESEC/FSE*, 2007.
- [8] R. Zhao, M. Harman, and Z. Li, “Empirical study on the efficiency of search based test generation for EFSM models,” in *3rd ICSTW*, 2010.
- [9] K. Lakhota, M. Harman, and H. Gross, “AUSTIN: An open source tool for search based software testing of C programs,” *Inf. Softw. Technol.*, vol. 55, no. 1, 2013.
- [10] A. Mehrmand and R. Feldt, “A factorial experiment on scalability of search-based software testing,” in *3rd AITSE*, 2010.
- [11] A. Arcuri, “Full theoretical runtime analysis of alternating variable method on the triangle classification problem,” in *1st SSBSE*, 2009.
- [12] J. Kempka, P. McMinn, and D. Sudholt, “Design and analysis of different alternating variable searches for search-based software testing,” *Theor. Comp. Sci.*, 2015, In Press.

Similarity-based regression test case prioritization

Rongcun Wang
School of Computer Science
and Technology
China University of Mining and
Technology
Xuzhou, 221116, China
Email:rcwang@hust.edu.cn

Shujuan Jiang
School of Computer Science
and Technology
China University of Mining and
Technology
Xuzhou, 221116, China
Email:shjjiang@cumt.edu.cn

Deng Chen
Hubei Provincial Key Laboratory of
Intelligent Robot
Wuhan Institute of Technology
Wuhan, 430073, China
Email:chendeng8899@hust.edu.cn

Abstract—With the continuous evolution of software systems, test suites often grow very large. Rerunning all test cases may be impractical in regression testing under limited resources. Coverage-based test case prioritization techniques have been proposed to improve the effectiveness of regression testing. The original test suite often contains some test cases which are designed for exercising production features or exceptional behaviors, rather than for code coverage. Therefore, coverage-based prioritization techniques do not always generate satisfactory results. In this context, we propose a global similarity-based regression test case prioritization approach. The approach reschedules the execution order of test cases based on the distances between pair-wise test cases. We designed and conducted empirical studies on four C programs to validate the effectiveness of our proposed approach. Moreover, we also empirically compared the effects of six similarity measures on the global similarity-based test case prioritization approach. Experimental results illustrate that the global similarity-based regression test case prioritization approach using Euclidean distance is the most effective. This study aims at providing practical guidelines for picking the appropriate similarity measures.

Keywords—regression testing, test case prioritization, similarity measures

I. INTRODUCTION

Regression testing is a very time-consuming and expensive activity. It accounts for more than 50% of the cost of software maintenance [1]. With the continuous evolution of software systems, test suites grow very large. The execution of the whole test suite consumes more time and resources. The high testing cost conflicts with constrained time and resources. Many test case prioritization techniques have been proposed to solve the contradiction [2], [3]. They aim at rescheduling the execution order of test cases to detect faults as early as possible.

Coverage-based prioritization techniques have gained wide attention. Most of these techniques resort to use greedy or metaheuristic search algorithms [5] to maximize the possible coverage. The original test suite often contains some test cases which are designed for exercising production features or exceptional behaviors, rather than for code coverage [6]. Therefore, coverage-based prioritization techniques do not always generate satisfactory results.

More recently, similarity-based test case prioritization techniques, incorporating clustering-based [7], ART-based [8] and

other similarity-based prioritization [9], have been developed. Similarity-based test case prioritization techniques assume that test case diversity aids to detect more faults[15], [13]. Clustering-based prioritization techniques assume that the test cases within the same cluster have the same fault detection capability. Clustering-based prioritization techniques significantly depend on the number of clusters. Similarly, ART-based prioritization technique selects a test case to be prioritized from a subset of all remaining test cases. In other words, the method does not assure global test case diversity. Therefore, we propose a global similarity-based test case prioritization approach. Our approach rearranges the execution order of test cases from the global perspective. More importantly, our proposed approach is nonparametric.

We empirically evaluate the effects of six similarity measures, including Jaccard Index (JI), Gower-Legendre (GL), Soka-Sneath (SS), Euclidean distance (ED), Cosine similarity (CS) [10], and Proportional distance (PD) [11]metric, on the global similarity-based prioritization algorithm over 4 programs written in the C language. One way variance analysis (ANOVAs) [12] is used to analyze the statistical difference between different similarity measures. The results illustrate that Euclidean distance is superior to other similarity measures in terms of fault detection rate and standard deviation. The global similarity-based prioritization algorithm using Euclidean distance outperforms random prioritization and the additional function coverage prioritization. Our proposed approach is comparable to the best coverage-based prioritization techniques, i.e., the additional branch coverage prioritization technique. This study provides a practical guide for picking the optimal similarity measures.

The rest of this paper is organized as follows: Section II describes our approach. Experimental design and results analysis are presented in Section III and Section IV. The threats to validity are discussed in Section V. Section VI summarizes related works. The conclusions are described in Section VII.

II. METHODOLOGY

A. Overview

Figure 1 summarizes similarity-based test case prioritization techniques, which mainly include four steps:

(1) Instrumentation

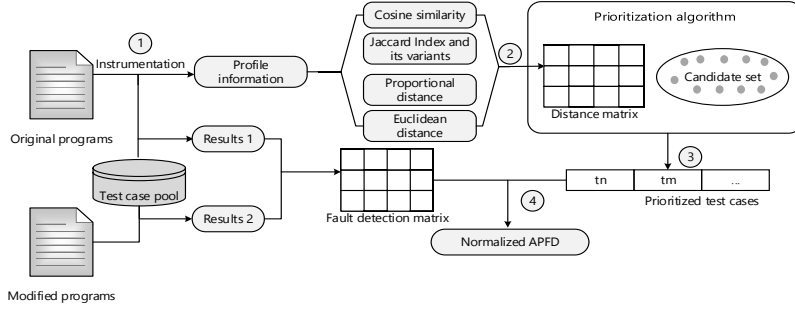


Fig. 1. Overview of similarity-based test case prioritization

With the dynamic instrumentation tool `gCOV`, we collect execution profiles and construct branch coverage vectors.

(2) Distance calculation

The distance between pair-wise test cases is calculated by a certain distance measure.

(3) Test case prioritization

Test cases are prioritized based on the distance between pair-wise test cases.

(4) Evaluation

The fault detection effectiveness of a prioritized test suite is evaluated based on the relation between faults and test cases.

B. Similarity measures

All branches covered by a test case, can be represented as a binary branch coverage vector $V: \langle v_1, v_2, \dots, v_n \rangle$, where v_i is 0 if the i th branch is covered, otherwise 1. Similarly, the vector can also be implemented with numeric entries, i.e., v_i represents the number of times that i th branch is executed.

1) *Cosine similarity*: The binary branch coverage vectors generated by executing test case t_1 and t_2 are $X: \langle x_1, x_2, \dots, x_n \rangle$ and $Y: \langle y_1, y_2, \dots, y_n \rangle$, respectively. The similarity between t_1 and t_2 is defined as follows:

$$s(t_1, t_2) = \frac{X^t \cdot Y}{\|X\| \|Y\|}, \quad (1)$$

where X^t is a transposition of vector X , and $\|X\|$ is the Euclidean norm of vector X . Similarly, $\|Y\|$ is the Euclidean norm of vector Y . In essence, s is the cosine of the angle between X and Y . For Cosine similarity, the corresponding dissimilarity is defined as $d(t_1, t_2) = 1 - s(t_1, t_2)$.

2) *Euclidean distance*: The Euclidean distance between test case t_1 and t_2 is defined as follows:

$$d(t_1, t_2) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}. \quad (2)$$

3) *Proportional distance*: Let $P: \langle p_1, p_2, \dots, p_n \rangle$ and $Q: \langle q_1, q_2, \dots, q_n \rangle$ stand for two coverage vectors implemented with numeric entries by executing t_1 and t_2 . The proportional distance between t_1 and t_2 is defined as follows:

$$d(t_1, t_2) = \sqrt{\sum_{i=1}^n \left(\frac{|p_i - q_i|}{\max_i - \min_i} \right)^2}, \quad (3)$$

where \max_i and \min_i represent the maximum and minimum number of times that the i th branch is executed over all tests in the test suite, respectively. When the difference between \max_i and \min_i is equal to 0, the corresponding term is also equal to 0.

4) *Jaccard Index and its variants*: The similarity between t_1 and t_2 based on Jaccard Index and its variants is defined as follows:

$$s(t_1, t_2) = \frac{X \cdot Y}{X \cdot Y + w(\|X\|^2 + \|Y\|^2 - 2(X \cdot Y))}, \quad (4)$$

where $X \cdot Y$ is the inner product of X and Y . When w is equal to 1, the above formula is called Jaccard Index. If $w=2$ and $1/2$, this formula is called Soka-Sneath measure and Gower-Legendre measure, respectively. For Jaccard Index and its variants, the corresponding distance is $d(t_1, t_2) = 1 - s(t_1, t_2)$.

C. Prioritization Algorithm

The global similarity-based test case prioritization algorithm (GSTCP) selects a test case from all not yet prioritized test cases, rather than a candidate set of them [8]. Its pseudo-code is described in Algorithm 1.

This algorithm randomly selects a test case t_k from the original test suite T , deletes it from T , and adds it to the tail of a prioritized sequence P . The distance from test case t in T to t_k is viewed as the minimal set distance from t to P . Function *dist* is responsible for the calculation of the distance between two test cases. The process of test case prioritization mainly includes two steps. The first step is to seek a test case t_i in T that has the maximum distance from the last test case in P (Line 9-Line 13). Test case t_i is added to the tail of P and deleted from T . The second step is to update the minimal set distance from test case t in T to P by comparing the distance from t to t_i and the set distance from t to P before adding t_i to P (Line 15-Line 19).

In the process of prioritizing test cases, this algorithm needs to calculate the distance between remaining test cases in T and the last test case in P . The number of times that this algorithm calculates the distance gradually decreases from $n-1$ to 1. The time complexity and space complexity of GSTCP are $O(n^2)$ and $O(n)$, respectively. Compared with the ART-based prioritization algorithm, GSTCP significantly reduces time complexity.

Algorithm 1: GSTCP

Input : A test suite $T : \{t_0, t_1, \dots, t_{n-1}\}$
Output: A prioritized sequence $P : \langle p_0, p_1, \dots, p_{n-1} \rangle$

```
1  $P \leftarrow \emptyset$ ; double  $dist\_min[n]$ ;  
2 randomly select test case  $t_k$  from  $T$ ;  
3  $max \leftarrow k$ ;  $T \leftarrow T \setminus \{t_{max}\}$ ;  $P \leftarrow \langle t_{max} \rangle$ ;  
4 for  $i \leftarrow 0$  to  $n - 1$  do  
5    $dist\_min[i] = dist(t_j, t_{max})$ ;  
6 end  
7 repeat  
8    $max\_dist = 0.0$ ;  
9   for  $i \leftarrow 0$  to  $n - 1$  do  
10    if  $dist\_min[i] > max\_dist$  and  $t_i \in T$  then  
11       $max\_dist = dist\_min[i]$ ;  $max \leftarrow i$ ;  
12    end  
13  end  
14  add  $t_{max}$  to the tail of  $P$ ;  $T \leftarrow T \setminus \{t_{max}\}$ ;  
15  for  $i \leftarrow 0$  to  $n - 1$  do  
16    if  $dist\_min[i] > dist(t_i, t_{max})$  and  $t_i \in T$  then  
17       $dist\_min[i] = dist(t_i, t_{max})$ ;  
18    end  
19  end  
20 until  $T$  is empty;  
21 return  $P$ 
```

III. EMPIRICAL STUDY

A. Research Questions

In the empirical study, we address the following two specific research questions.

RQ1: Do different similarity measures have significant effects on the global similarity-based test case prioritization algorithm?

RQ2: Can the most effective global similarity-based prioritization technique be as effective as coverage-based prioritization techniques?

B. Subject Program

Our experiments were conducted over four subject programs¹, incorporating 2 small-sized Siemens programs and 2 medium-sized UNIX utilities. Descriptive information about the selected programs is presented in Table I, where the lines of codes are calculated by the tool “SLOCCount”².

TABLE I
DESCRIPTIVE INFORMATION FOR SUBJECT PROGRAMS

Program Name	Fault Versions	Lines of Code	Test Suite Size
print_tokens	7	341-343	4130
print_tokens2	10	350-355	4115
make	33	12609-17153	1043
sed	18	4711-9204	370

¹<http://sir.unl.edu/php/showfiles.php>

²<http://www.dwheeler.com/sloccount>

We eliminated the fault versions whose faults cannot be detected by any test case. Likewise, if a fault can be detected by more than 20% of test cases, the fault is also excluded.

C. Evaluation Metric

The average percentage of faults detected (*APFD*) [18] is commonly used to evaluate the effectiveness of a prioritization technique implemented on a whole test suite. Let T be a test suite containing n test cases and let F be a set of m faults exposed by T . *APFD* is defined as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}, \quad (5)$$

where TF_i is the first test case in a prioritized test suite that detects fault i .

The application of *APFD* assumes that the whole test suite can be run and find all of the faults. Only a part of test suite is often executed in regression testing [14], [19]. In this sense, *APFD* is unsuitable to measure the effectiveness of a prioritization technique implemented on a part of test suite. Therefore, we use a metric, called Normalized *APFD* [20], which utilizes information on both fault finding and time of detection. Let TF_i be the first test case in the prioritized and reduced test suite T' of T that detects fault i . Let n' be the size of T' . Normalized *APFD* is defined as follows:

$$NAPFD = p - \frac{TF_1 + TF_2 + \dots + TF_m}{n'm} + \frac{p}{2n'}, \quad (6)$$

where p represents the quotient of the number of faults detected by the prioritized and reduced test suite divided by the number of faults detected in the whole test suite. If a fault i is never detected by T' , TF_i is set 0.

Since all prioritization algorithms in this study have the nature of randomness, we repeated 100 times for every prioritization algorithm with different random seeds. The mean *NAPFD* values of every sample were calculated.

IV. EXPERIMENTAL RESULTS

We took 10 samples starting from 2% to 20% with the increment of 2% for every program so as to look deeper into the statistical difference between different similarity measures.

A. Experiment 1

1) *Experimental Results:* The experimental results are shown in Figure 2, where the x-axis of each graph indicates the number of tests selected; while the y-axis shows the mean *NAPFD* of each similarity measure.

From Figure 2, Cosine similarity performed better or as good as Jaccard Index in terms of *NAPFD*. Likewise, Euclidean distance also performed consistently better than Jaccard Index for smaller samples. Particularly, Euclidean distance outperformed other similarity measures for all samples over the program *print_tokens* and *print_tokens2*. The results of Jaccard Index were very close to those of its variants.

Table II summarizes the standard deviations for all similarity measures. The minimal standard deviations are reported highlighting cells in gray shade. From Table II, Euclidean distance

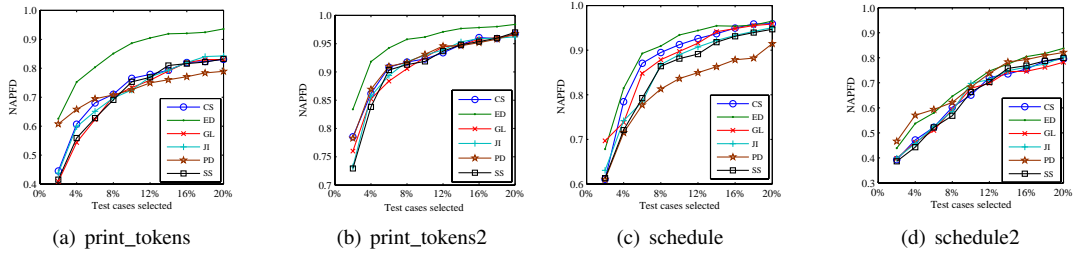


Fig. 2. NAPFDs of six similarity measures using the global similarity-based test case prioritization algorithm

TABLE II
THE STANDARD DEVIATIONS OF DIFFERENT SIMILARITY MEASURES BASED ON THE GLOBAL SIMILARITY PRIORITIZATION ALGORITHM

Program	SM	Sampling proportion									
		2%	4%	6%	8%	10%	12%	14%	16%	18%	20%
print_tokens	CS	0.145	0.125	0.100	0.116	0.084	0.086	0.065	0.070	0.070	0.079
	ED	0.098	0.085	0.083	0.062	0.055	0.045	0.043	0.037	0.039	0.035
	PD	0.121	0.109	0.113	0.109	0.110	0.109	0.125	0.106	0.108	0.109
	JI	0.136	0.112	0.108	0.093	0.099	0.086	0.068	0.067	0.078	0.074
	GL	0.117	0.137	0.105	0.108	0.094	0.083	0.085	0.077	0.075	0.065
	SS	0.145	0.122	0.111	0.094	0.099	0.087	0.089	0.077	0.073	0.069
print_tokens2	CS	0.099	0.065	0.050	0.050	0.036	0.034	0.029	0.020	0.024	0.019
	ED	0.066	0.036	0.027	0.025	0.015	0.013	0.012	0.009	0.008	0.007
	PD	0.139	0.079	0.059	0.060	0.051	0.032	0.039	0.039	0.029	0.031
	JI	0.109	0.070	0.050	0.040	0.030	0.035	0.023	0.023	0.022	0.023
	GL	0.094	0.058	0.059	0.051	0.039	0.029	0.030	0.021	0.016	0.015
	SS	0.097	0.080	0.048	0.044	0.044	0.033	0.028	0.026	0.024	0.017
make	CS	0.277	0.180	0.099	0.084	0.057	0.055	0.045	0.033	0.031	0.024
	ED	0.167	0.133	0.079	0.058	0.046	0.051	0.036	0.035	0.046	0.025
	PD	0.280	0.265	0.220	0.241	0.233	0.177	0.158	0.196	0.156	0.116
	JI	0.285	0.239	0.240	0.132	0.172	0.122	0.118	0.145	0.049	0.074
	GL	0.213	0.204	0.178	0.137	0.138	0.193	0.053	0.096	0.049	0.068
	SS	0.269	0.253	0.189	0.124	0.131	0.152	0.136	0.113	0.076	0.070
sed	CS	0.055	0.064	0.056	0.055	0.053	0.054	0.048	0.040	0.038	0.042
	ED	0.038	0.042	0.040	0.031	0.042	0.033	0.031	0.035	0.026	0.032
	PD	0.062	0.540	0.054	0.041	0.043	0.048	0.039	0.406	0.034	0.043
	JI	0.062	0.052	0.055	0.053	0.041	0.045	0.048	0.051	0.043	0.041
	GL	0.058	0.056	0.043	0.060	0.042	0.052	0.051	0.052	0.047	0.041
	SS	0.051	0.061	0.050	0.053	0.051	0.058	0.038	0.043	0.044	0.040

generated smaller standard deviations than other similarity measures with the same sampling proportion. This means that the application of Euclidean distance reduces the risk of missing faults in practice.

2) *Experimental Analysis*: We conducted ANOVAs to verify whether different similarity measures generate the significant effects on the global similarity-based prioritization algorithm at 5% significance level. Having executed ANOVAs, we find that the p -value was less than 0.05 for every sample across every subject program. In other words, different measures have significant effects on the global similarity-based prioritization algorithm. We further conducted multiple comparisons so as to seek which similarity measures can produce higher NAPFDs.

Table III shows the results of multiple comparisons between pair-wise similarity measures, where each element (m, n) denotes the “win/tie/loss” (win: the number of times that the m -th row measure performs significantly better than the n -th column measure; tie: the number of times that there is no significant difference between the m -th row measure and the

n -th column measure; loss: the number of times that the m -th row measure performs significantly worse than the n -th column measure) value.

TABLE III
THE SUMMARIZED RESULTS OF MULTIPLE COMPARISONS BETWEEN PAIR-WISE SIMILARITY MEASURES OVER 40 DIFFERENT EXPERIMENTAL SETTINGS (4 PROGRAMS \times 10 SAMPLING RATES)

	CS	ED	GL	JI	PD
ED	(27/12/1)	-	-	-	-
GL	(1/31/8)	(1/9/30)	-	-	-
JI	(1/32/7)	(1/8/31)	(2/34/4)	-	-
PD	(6/20/14)	(2/9/29)	(8/19/13)	(9/19/12)	-
SS	(0/32/8)	(0/8/32)	(1/36/3)	(3/35/2)	(12/20/8)

From Table III, Euclidean distance performed better than other similarity measures. In the best case, Euclidean distance outperformed Soka-Sneath on 32 settings, while Soka-Sneath did not outperformed it on any setting. In the worst case, Euclidean distance outperformed Cosine similarity on 27 settings,

while Cosine similarity outperformed it only on 1 settings. The results generated by Jaccard Index were very close to those of its variants. The plausible explanation is that the topologies of Jaccard Index and its variants are very similar.

B. Experiment 2

1) *Experimental Results*: We mainly compared the global similarity-based prioritization using Euclidean distance with random prioritization (RP), the additional function coverage prioritization (AF), the additional statement coverage prioritization (AS), and the additional branch coverage prioritization (AB) [4]. Figure 3 shows the *NAPFD*s of different prioritization techniques. Table IV summarizes the standard deviations for different prioritization techniques.

TABLE V
THE SUMMARIZED RESULTS OF MULTIPLE COMPARISONS BETWEEN DIFFERENT PRIORITIZATION TECHNIQUES OVER 40 DIFFERENT EXPERIMENTAL SETTINGS (4 PROGRAMS \times 10 SAMPLING RATES)

	RP	AF	AS	AB
AF	(24/16/0)	-	-	-
AS	(40/0/0)	(22/13/5)	-	-
AB	(40/1/0)	(25/9/6)	(14/26/0)	-
ED	(39/1/0)	(23/14/3)	(10/22/8)	(10/18/12)

2) *Experimental Analysis*: Table V shows the results of multiple comparisons between different prioritization algorithms. Our approach performed significantly better than RP, and AF in terms of *NAPFD*. In general, the global similarity-based prioritization algorithm using Euclidean distance was equal to AS. ED outperformed AS on 10 settings, while AS outperformed it on 8 settings. Particularly, our proposed approach was comparable to the best coverage-based prioritization algorithm, i.e., AB. ED outperformed AB on 10 settings, while AB outperformed it on 12 settings.

Additionally, our proposed approach is superior to other prioritization techniques with respect to the standard deviation of *NAPFD*. This means that it can yield more reliable results. In summary, the global similarity-based prioritization algorithm using Euclidean distance is very promising as a candidate for regression test case prioritization.

V. THREATS TO VALIDITY

This section discusses the potential threats to validity.

Threats to internal validity are from the effects of instrumentation and the sampling rates. Therefore, we collected profile information using a professional tool *gCOV*. Additionally, we took 10 samples for every subject program and reported the mean value of every sample.

Threats to external validity for this study concern the representativeness of the programs utilized. Although the four programs are from different domains with different characteristics, they may not be representative of all other programs. The threat can be addressed by selecting larger scale and more representative industrial programs in future work.

Threats to construct validity may be affected by the evaluation metric. As previous studies, *NAPFD* is used to evaluate

the effectiveness of different prioritization techniques. This may be insufficient for evaluating the effectiveness of different combinations. There may be other metrics which are more relevant to this study.

VI. RELATED WORK

Ledru *et al.* used string distances for test case prioritization [9]. They empirically evaluated the effects of string edit distances on test case prioritization. Test cases were ordered before executing them. Similarly, Hemmati *et al.* introduced a family of similarity-based test case selection techniques for model-based testing [13], [15]. The above two methods were applied to black-box testing. On the contrary, our approach uses dynamic profile information generated by executing test cases to reschedule the execution order.

Clustering algorithms have been also applied to test case prioritization. Yoo *et al.* [7] combined clustering with expert knowledge to achieve scalable prioritization. The process of prioritization depended on human efforts, which is different from ours. Dickinson *et al.* [11] presented distribution-based filtering and prioritizing techniques incorporating sampling methods. Clustering-based prioritization techniques significantly depend on a parameter, i.e., the number of clusters. On the contrary, our approach is nonparametric, i.e., it does not depend on the number of clusters.

More recently, similarity-based algorithms have been applied to regression test case prioritization. Krishna *et al.* [16] used Levenshtein distance to similarity-based test prioritization. Jiang *et al.* [8] proposed a new family of coverage-based ART techniques and used Jaccard Index to measure the distance between pair-wise test cases. Fang *et al.* [17] introduced several similarity-based test case prioritization techniques based on the edit distances of ordered sequences. However, we represented the execution profile that a test case exercised into a branch coverage vector, rather than an ordered sequence [17]. Differently from Jiang *et al.* [8] and Krishna *et al.* [16], where only one similarity measure was used, we empirically evaluated six similarity measures.

VII. CONCLUSIONS

In this paper, we proposed a global similarity-based test case prioritization algorithm based on the comparison of similarity measures between test cases in a given test case suite. By ANOVAs, we find that different measures have significant effects on the global similarity-based test case prioritization algorithm. Particularly, Euclidean distance perform better than other five similarity measures in terms of *NAPFD* and standard deviation. Therefore, we recommend Euclidean distance. Moreover, the global similarity-based prioritization algorithm using Euclidean distance outperforms random prioritization and the additional function coverage prioritization with respect to *NAPFD*. It is comparable to the additional branch coverage. Since our proposed approach generates smaller standard deviation, it can provide more reliable results.

For future work, much more similarity measures will be empirically evaluated. Least but not last, we will collect more

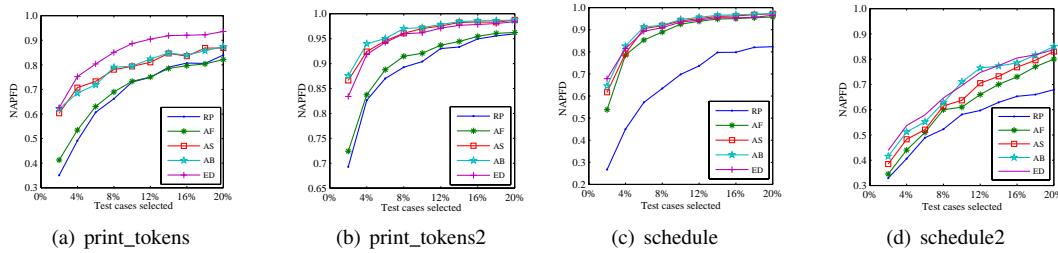


Fig. 3. NAPFDs of different test case prioritization techniques

TABLE IV
THE STANDARD DEVIATIONS OF DIFFERENT PRIORITIZATION ALGORITHMS

Program	TCP	Sampling proportion									
		2%	4%	6%	8%	10%	12%	14%	16%	18%	20%
print_tokens	RP	0.155	0.149	0.135	0.126	0.113	0.105	0.095	0.089	0.087	0.084
	AF	0.141	0.136	0.120	0.101	0.102	0.085	0.080	0.087	0.077	0.069
	AS	0.128	0.114	0.097	0.084	0.092	0.085	0.074	0.076	0.077	0.062
	AB	0.112	0.106	0.090	0.087	0.085	0.077	0.079	0.076	0.069	0.063
	ED	0.098	0.085	0.083	0.062	0.055	0.045	0.043	0.037	0.039	0.035
print_tokens2	RP	0.138	0.123	0.119	0.095	0.087	0.086	0.084	0.073	0.074	0.065
	AF	0.130	0.107	0.098	0.083	0.074	0.077	0.052	0.048	0.044	0.039
	AS	0.085	0.079	0.072	0.070	0.063	0.057	0.056	0.041	0.031	0.009
	AB	0.075	0.068	0.054	0.057	0.049	0.036	0.030	0.021	0.020	0.008
	ED	0.066	0.036	0.027	0.025	0.015	0.013	0.012	0.009	0.008	0.007
make	RP	0.260	0.225	0.194	0.156	0.121	0.116	0.105	0.096	0.073	0.082
	AF	0.236	0.173	0.132	0.107	0.091	0.079	0.052	0.036	0.025	0.022
	AS	0.190	0.141	0.093	0.080	0.055	0.035	0.049	0.026	0.013	0.011
	AB	0.172	0.152	0.095	0.072	0.053	0.034	0.038	0.039	0.012	0.015
	ED	0.167	0.133	0.079	0.058	0.046	0.051	0.036	0.032	0.046	0.025
sed	RP	0.126	0.113	0.095	0.097	0.073	0.052	0.061	0.045	0.038	0.029
	AF	0.126	0.090	0.072	0.063	0.044	0.040	0.032	0.030	0.021	0.020
	AS	0.086	0.073	0.069	0.043	0.024	0.022	0.035	0.017	0.025	0.021
	AB	0.072	0.065	0.052	0.030	0.027	0.021	0.022	0.018	0.015	0.017
	ED	0.038	0.042	0.043	0.040	0.031	0.042	0.033	0.031	0.026	0.032

coverage information test cases exercised and construct differently structural profiles. The effects of differently structural profiles on test case prioritization will also be empirically evaluated.

REFERENCES

- [1] M. J. Harrold and A. Orso. Retesting software during development and maintenance. In *FOSM'08*, pp. 99-108, 2008.
- [2] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *ISSRE'97*, pp. 230-238, 1997.
- [3] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Test case prioritization: an empirical study. In *ICSM'99*, pp. 179-188, 1999.
- [4] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: a family of empirical studies. *IEEE Trans. Softw. Eng.*, 28(2):159-182, 2002.
- [5] Z. Li, M. Harman, R. M. Hierons. Search algorithms for regression test prioritization. *IEEE Trans. Softw. Eng.*, 33(4):225-237, 2003.
- [6] G. Rothermel, M. J. Harrold, J. V. Ronne, and C. Hong. Empirical studies of test suite reduction. *Softw. Test. Verif. Rel.*, 12: 219-249, 2002.
- [7] S. Yoo, M. Harman, P. Tonella, and A. Susi. Clustering test cases to achieve effective and scalable prioritization incorporating expert knowledge. In *ISSTA'09*, pp. 201-212, 2009.
- [8] B. Jiang, Z. Y. Zhang, W. K. Chan, and T. H. Tse. Adaptive random test case prioritization. In *ASE'09*, pp. 233-244, 2009.
- [9] Y. Ledru, A. Petrenko, and S. Borody. Prioritizing test cases with string distances. *Automat. Softw. Eng.*, 19(1):65-95, 2012.
- [10] R. Xu and D. Wunsch. A survey of clustering algorithms. *IEEE Trans. Neural Netw.*, 16(3):645-678, 2005.
- [11] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *ICSE'01*, pp. 339-348, 2001.
- [12] H. Scheffe. *The Analysis of Variance*. John Wiley and Sons, New York, 1993.
- [13] H. Hemmati, A. Arcuri, and L. Briand. Achieving scalable model-based testing through test case diversity. *ACM Trans. Softw. Eng. Methodol.*, 22(1):1-42, 2013.
- [14] J. Jones, and M. Harrold. Test suite reduction and prioritization for modified condition/decision coverage. *IEEE Trans. Softw. Eng.*, 29(3):193-209, 2003.
- [15] H. Hemmati and L. Briand. An industrial investigation of similarity measures for model-based test case selection. In *ISSRE'10*, pp. 141-150, 2010.
- [16] M. Krishna, M. Koyuturk, A. Grama, and S. Jagannathan. PHALANX: A graph-theoretic framework for test case prioritization. In *SAC'08*, pp. 667-673, 2008.
- [17] C. Fang, Z. Chen, K. Wu, Z. Zhao. Similarity-based test case prioritization using ordered sequences of program entities. *Softw. Quality J.*, 22(2):335-361, 2014.
- [18] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. Softw. Eng.*, 27(10):929-948, 2001.
- [19] A. Smith, J. Geiger, G. Kapfhammer, and M. Soffa. Test suite reduction and prioritization with call trees. In *ASE'07*, pp. 539-540, 2007.
- [20] X. Qu, M. B. Cohen, and K. M. Wolf. Combinatorial interaction regression testing: a study of test case generation and prioritization. In *ICSM'07*, pp. 255-264, 2007.

Secure, Dynamic and Distributed Access Control Stack for Database Applications

Óscar Mortágua Pereira¹, Diogo Domingues Regateiro², Rui L. Aguiar³

Instituto de Telecomunicações
DETI, University of Aveiro
Aveiro, Portugal
{omp¹, diogoregateiro², ruilaa³}@ua.pt

Abstract— In database applications, access control security layers are mostly developed from tools provided by vendors of database management systems and deployed in the same servers containing the data to be protected. This solution conveys several drawbacks. Among them we emphasize: 1) if policies are complex, their enforcement can lead to performance decay of database servers; 2) when modifications in the established policies implies modifications in the business logic (usually deployed at the client-side), there is no other possibility than modify the business logic in advance and, finally, 3) malicious users can issue CRUD expressions systematically against the DBMS expecting to identify any security gap. In order to overcome these drawbacks, in this paper we propose an access control stack characterized by: most of the mechanisms are deployed at the client-side; whenever security policies evolve, the security mechanisms are automatically updated at runtime and, finally, client-side applications do not handle CRUD expressions directly. We also present an implementation of the proposed stack to prove its feasibility. This paper presents a new approach to enforce access control in database applications, this way expecting to contribute positively to the state of the art in the field.

Keywords— information security; access control; database; SQL; software architecture.

I. INTRODUCTION

Access control [1], [2] is a critical security issue in many software systems. Access control “*is concerned with limiting the activity of legitimate users.*” [3]. Basically, it is a process to supervise every user’s requests to access protected resources, in our case data residing inside database management systems (DBMS), by determining whether authorizations should be granted or denied. Access to data stored in DBMS is mostly achieved issuing Create, Read, Update and Delete (CRUD) expressions from client-side applications. Nevertheless, access control security layers are traditionally deployed in centralized servers where the data to be protected are stored. Rephrasing, the attackers are distributed and the security wall is built around the data. Very few scientific contributions have been proposed based on a distributed architecture. Basically, in a distributed architecture the walls are not centralized but deployed by putting them close to the attackers, as in [4], [5]. At first sight, the distributed architecture would seem the best one. Although, distributed architectures raise additional security concerns, such as how to trust that legitimate users are

issuing authorized requests only. To overcome this security concern, in this paper we propose a security stack, herein referred to as Secure, Dynamic and Distributed Access Control Stack for Database Applications. The security stack comprises the policies to be enforced and also the correspondent authorized Create, Read, Update and Delete (CRUD) expressions. In order to keep access control mechanisms always aligned with the established policies, they are automatically updated at runtime whenever policies are modified. We also provide an empirical proof of concept to demonstrate the feasibility of the proposed security stack. It is expected that the outcome of this paper can contribute to new approaches to enforce access control policies, namely by deploying them based on distributed architectures.

The remainder of this paper is organized as hereafter described. Chapter II presents the motivation for the work detailed in this paper, chapter III presents the related work, chapter IV presents the security stack, chapter V presents a proof a concept of the security stack and chapter VI concludes this paper and details the future work.

II. MOTIVATION

Currently, there is no standard to enforce access control policies in database applications. Security experts complement security layers built from embedded tools provided by vendors of DBMS with additional security artifacts (components and/or hard-coded inside the client-application) built from scratch and tailored to the specific scenario. Approaches based on this approach convey several drawbacks. Among them we emphasize:

Scalability: security layers based on DBMS tools share concurrently the computational resources allocated to DBMS. When access control policies are many and complex, they can lead to performance decay. Moreover, very often the security layers resort to additional techniques, such query rewriting techniques [6][7][8][9][10], the use of views [10][11][12][11][12][6] and parameterized views [15]. Inevitably, these additional artifacts will lead to the allocation of additional computational resources and, therefore, additional performance decay, as explicitly recognized in most of the papers by their authors.

Maintainability: security layers built from DBMS tools can be easily maintained since they are centralized in a server. In

opposite, security artifacts deployed in client-side applications can lead to huge maintenance efforts. For example, if an attribute of a table becomes protected by a new or modified policy, there is no other alternative than modify in advance all the CRUD expressions violating that policy. Otherwise, CRUD expressions are rejected while the updating process is not completed.

Security gap: CRUD expressions are mostly issued from client-side applications. If this process is not controlled, malicious users can issue CRUD expressions systematically against the DBMS expecting to identify any security gap. It is very likely that professional malicious users will end up violating the protected data to some extension. Malicious users can follow additional malicious approaches, such as SQL Injection, the use of sequences of valid CRUD expressions [16] and also resorting to reflection [17] to get and/or modify the way software works. Other security gaps can arise from collecting data sent between a legitimate user and the DBMS and also from personification.

To overcome these drawbacks, we propose a security stack with the following properties: 1) access control mechanisms are mostly distributed in each client-side system; 2) access control mechanisms are updated whenever modifications occur in the established policies; 3) client-side applications are prevented from issuing CRUD expressions and, finally, 4) additional security mechanisms are provided for user authentication and secure connections.

III. RELATED WORK

We will now discuss the work in the field of access control enforcement and how they relate to our proposed security stack.

A complete architecture for web applications is presented in [18], where the problem of sensitive data being stored in a browser is solved by enforcing end-to-end security on data, across the virtual machine, operating system, networking and application layers. However, it relies solely on mandatory access control to enforce the end-to-end security policies and it is only concerned with web applications.

In [19] a new tool is presented, Ur/Web, where the access control policies can be checked by CRUD expressions written by programmers in a RDBMS backed system. It uses an extension to the standard SQL language with predicates that indicates 'which secrets the users knows' and determine what information can be disclosed. However, these predicates are not checked against the access control policies, potentially leaking protected information. λ_{DB} [20] is a programming language that enforces access control policies to data by static typing for data-centric programs. It allows the definition of entities that are checked at compile-time with the defined access control policies. Each entity has a set of attributes that are given a read and write permissions with different predicates, similarly to Ur/Web. Another similar work is presented in [21] using predicated grants. These solutions only provide access control mechanisms, not addressing the rest of the stack.

The work presented in [22] aims to provide role-based access control using proxy objects, generated using a custom compilation tool. Each role has a set of different proxy objects,

which are made available through Java Remote Method Invocation (RMI). Note that the proxy objects only implement the methods that the given role can execute, therefore it is not possible for a client to execute other methods. This method guards against reflection mechanisms since they do not work over sockets, but this solution is limited to RBAC and it protects access to java objects, not the data layer itself. Similarly, [23] is a security-typed programming language that extends Java that aims to give support for information flow and access control, enforced at both compile and runtime. However, this solution is also mostly used to manage information at the application level, leaving out other data sources, like a RDBMS.

In [4], the authors present a proposal to extend the RBAC model to control sequences of CRUD expressions. In [5], the authors present a proposal to implement distributed RBAC mechanisms. The content of both is relevant but they are focused on RBAC policies only. Additionally, key issues such avoiding the use of CRUD expressions at the client-side is also not supported.

IV. ACCESS CONTROL STACK PRESENTATION

For a software solution to provide secure, dynamic and distributed access control mechanisms we need to evaluate the requirements and the problems that arise from this architecture.

A. Access Control Stack

We will now discuss the necessary requirements to build Secure, Dynamic and Distributed Access Control Stack for Database Applications. As previously described, the identified fragilities of current solutions, and therefore to be addressed by the stack, are: scalability, maintainability and security gap. Figure 1 presents the general access control stack.

The data layer will obviously reside on the server side, so that it can be provisioned to all the clients. It can use relational DBMS[24] or some other form of data storage, e.g. a distributed file system[25][26] (e.g. Apache Hadoop[27]).

The application layer requesting access to the protected data resides on the client-side. It will access the data layer through the Security Layer.

The Security Layer is the layer responsible to ensure that all operations requested by the Application layer follow the established security rules. It comprises three main components: Security Manager, Access Control and Network Security.

Security Manager: The Security Manager component needs to address three main issues. The first one is to ensure that access control mechanisms are dynamically built and updated at runtime. This is very important to overcome one of the fragilities of current solutions: maintainability. The building and updating processes need to be based on an automated engine responsible for generating the necessary code for the Access Control component. The automated engine takes as input the policies to be enforced and also the architectural model to be implemented, as shown in Figure 2. The place to store the policies to be enforced depends on each particular use case or scenario. The second one is ensure that the access to the Data layer follows the established security rules. The third and last one is provide a standard interface to Application layer.

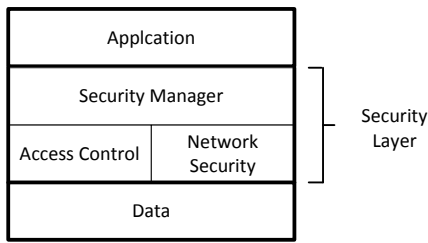


Figure 1. General access control stack.

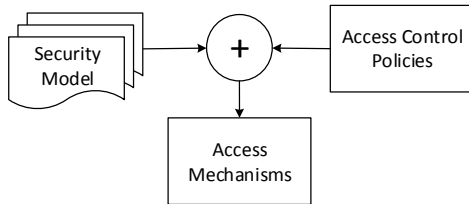


Figure 2. Access control mechanisms generation process.

This is quite relevant because Access Control component is not static as previously described. It depends on the policies to be enforced.

Network Security: Connections to database need to follow some security rules, namely authentication and secure connections. These rules are enforced by the Network Security component. Authentication mechanism forces users to present some sort of identification before accessing the data. To prevent malicious users from accessing the data when it is sent to the authenticated users it should be possible to setup a secure communication channel, which is typically done using the SSL/TLS [28] communication protocols.

Access Control: The architecture and functionalities of the Access Control component depend on the policy to be enforced and also on the architectural model to be implemented. Nevertheless, independently from the policy to be adopted, the drawbacks related to security gaps need to be addressed. Three main functionalities are required, as shown in Figure 3. The first one is the type of policy to be used, such as RBAC. The second one is the use of Sequence Controllers which are responsible to only allow the execution of valid sequences of CRUD expressions. The third one is the deployment of CRUD pointers instead of CRUD expressions. The first functionality enforces the top level functionalities of the policy to be enforced. The second functionality prevents malicious users from issuing sequences of authorized CRUD expressions to disclose the protected data. The third functionality prevents malicious users from resorting to techniques based on CRUD expressions to violate the established policies.

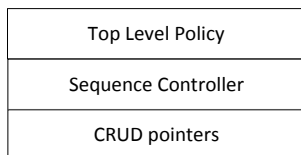


Figure 3. Access Control sub-components.

B. Technical Issues

Like any other software solution, there are technical issues that need to be solved before it can be expected to be considered useful. As such, we will now discuss the issues related to the Security Layer. However, it is impossible to discuss every issue that can rise from every possible scenario, so this chapter should be used as a starting point and the scenario-specific issues analyzed separately.

The Security Layer handles the Security Manager, the Access Control and the Network Security components. The Network Security aims at addressing two security problems: the network communication and the client authentication. The access control layer is where the access control effectively happens. It must be dynamic and distributed, which means that it must adapt to policy changes and be enforced on the client side. The Security Manager configures and manages the Network Security and the access control mechanisms.

Regarding the access control, the first problem comes from the fact that it is distributed. This means that the access control mechanisms will be subject to all kinds of attacks by malicious users that cannot be detected, since it happens on the client side. Therefore, there must be some mechanisms in place in order to prevent them. One of the major concerns are the reflection mechanisms[17] that some programming languages provide. These allow a program to inspect and modify the structures and behavior of the program at runtime (specifically the values, meta-data, properties and functions). This means that a software solution based on this stack cannot blindly rely on the distributed access control mechanisms to stop malicious users. To address this problem the access mechanisms were provided with CRUD pointers. CRUD pointers are some identifying tokens that are used by the client application instead of the actual CRUD expressions, which were pushed to the server-side. By making these pointers hard to guess and valid for a finite period of time, which must be small enough to prevent other users from using it by guessing, the usage of reflection mechanisms no longer threatens to manipulate the CRUD pointers.

The dynamic counterpart of the access control layer requires that the access control mechanisms in the client applications change as the access control policies change. This requires the clients to be notified somehow when they change and to enforce the changes immediately. To achieve this we have a Security Manager that implements the access control mechanisms using a security model that defines how the access control mechanisms should be created from the access control policies. It also provides them with the CRUD pointers received from the server and handles the network security procedures on behalf of the client application.

Finally, the network security has many problems to address, of which we will emphasize two: the problem that the data sent between two entities in a network can be read by anyone if no measures are taken to prevent it, and the problem that the entities involved generally do not prove their identity, allowing impersonation to occur. The first problem does not make it possible to the malicious user to manipulate the data being sent. However, sensible data (e.g. a client's identification) can still be acquired, which poses a serious security breach. The second

problem not only allows the data that is sent to be manipulated but also has all the problems the first problem implies, making it a greater concern. A common approach to handle this is the usage of SSL/TLS protocols, which can create a secure communication channel over an insecure medium, like the internet. SSL/TLS protocols verify the identity of servers and optionally also of users. Authentication of users is usually handled by a server side application that receives the client's identification tokens (e.g. a username and password) and provides the client application with the means to access the data if the tokens are valid.

V. PROOF OF CONCEPT

With this proof of concept we intend to demonstrate that the proposed access control stack is feasible of getting implemented. To achieve this we used a Java application as the client and the SQL Server 2010 RDBMS to manage the sample data we used. The implementation is called S-DRACA, which stands for Secure, Dynamic and Distributed Role-based Access Control Architecture.

A. Overview

We will now give an overview of the several components that are part of S-DRACA.

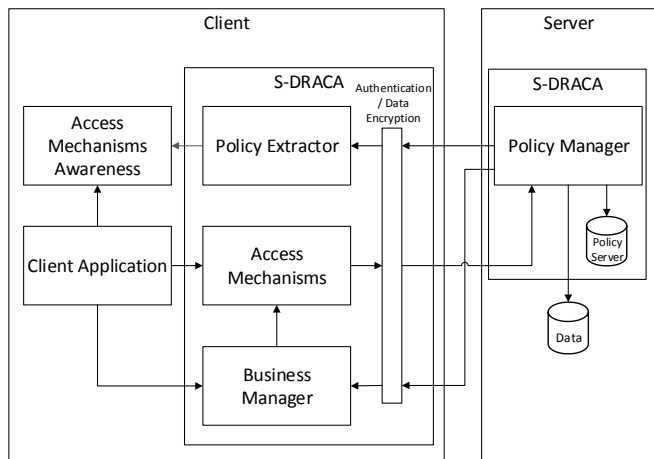


Figure 4. S-DRACA block diagram.

Figure 4 shows the block diagram of our proof of concept. We can see that on the server side we store the data in a database, along with the information about the access control policies in place on the system, which was stored in the Policy Server database. There is also a server application, the Policy Manager, which implements the SSL/TLS encryption and authentication mechanisms and also manages the policies to enforce in the system. In our proof of concept we only had one instance of a Policy Manager, but there could be many over different servers to distribute load in a cluster. On the client side we have the access mechanisms, which the client application uses to access the data stored on the server. The access mechanisms are instantiated by a Business Manager, which follows the policies defined on the server. We also have a Policy Extractor, a custom java annotation, which creates interfaces that the client application can use to access the data,

using the Security Model and the defined access control policies.

The access control policy used in our proof of concept is the role-based access control (RBAC). We chose this type of access control because it is natively supported by many DBMS, but the model used in the Policy Server could be changed to implement any type of access control policy without any implication to our proof of concept.

We will now explain how each layer was implemented in S-DRACA.

B. Layer Implementation

In S-DRACA, each layer was implemented independently for each other. The data layer is managed by the SQL Server 2010 RDBMS and it is accessed by the Policy Manager. It stores the data being protected as well as the access control policies defined for the system. Any operation requested by the client-application to manipulate the defined policies must go through the Policy Manager, to which the client application must be connected and authenticated. The requests to access the data are sent directly to the RDBMS, reusing the secure communication channel created when the client application connected to the Policy Manager to authenticate.

The network security component is implemented both on the client (Authentication and Data Encryption block), and on the server side (Policy Manager block). It uses several standards of the industry for data encryption and authentication, such as SSL/TLS and using hashed and salted passwords to store the client's credentials, respectively.

For the remainder of the security layer, we had to resolve the problems that originated from the programming language's reflection features and make it adapt dynamically to changes made to the policies defined in the system. Our access mechanisms are Java classes, called Business Schemas, see Figure 5, which only implement functions to access and manipulate data that the client application is allowed to. This is possible because when the client application authenticates with the Policy Manager, it receives the policies stored in the Policy Server that applies to said client. The Business Manager then

```

100  S_Cust = ss.businessService(Role_IRole_B1.s_customers,
101  Role_IRole_B1.s_customers_S_Customers_byCountry);
102  S_Cust.execute(country);
103  if(S_Cust.moveToNext()) {
104  custName= S_Cust.CustomerName();
105  // read more attributes
106  // ... code
107  S_Cust.beginUpdate();
108  S_Cust.uCustomerName(custName);
109  // update more attributes
110  S_Cust.updateRow();
111  // ... code
112  S_Cust.beginInsert();
113  S_Cust.iCustomerName(custName);
114  // insert more attributes
115  S_Cust.endInsert(true);
116  // ... code
117  S_Cust.deleteRow();
118  }

```

Figure 5. S-DRACA usage example.

uses this information and the Security Model to generate the Business Schemas with the appropriate methods (see Figure 2). The client application can then use these runtime generated Business Schemas because they implement interfaces generated by the Policy Extractor during compilation, known as Access Mechanisms Awareness, which also follow the Security Model. This prevents the clients from requesting operations that they do not have access to, but reflection mechanisms can still expose the private connection objects the Business Schemas use to request the operations. We have solved this issue using the CRUD pointers approach. Figure 5 shows a simple example of the core interface S-DRACA provides to the developers. The Business Schema `S_Customers` is instantiated (line 100) and executed (line 102) to obtain the data from the database. Then, if some data is returned, information can be read (line 104), updated (lines 107-110), inserted (lines 112-115) and deleted (line 117).

Note that not every user might see all the operations as shown in Figure 5. Since the Business Schemas are created dynamically from the access control policies retrieved from the server, only the authorized operations are actually implemented. This prevents developers from performing an operation they are not allowed to, which would only be known at runtime and could even go unnoticed for a long time if those operations are issued under rare circumstances.

To guarantee that the access control mechanisms adapt to changes made to the defined access control policies, the database notifies the Policy Manager, through the use of triggers, when and what information in the Policy Server was inserted, deleted or altered. This prompts the Policy Manager to verify which clients must be notified of the changes and send them the modifications. The Policy Manager of each client, upon receiving the changes, re-implements the Business Schemas and loads them, which takes immediate effect on current and future instantiations. However, the interfaces created by the Policy Server cannot be modified, since they were created during compilation, so until the client application is updated it will generate errors when methods that are no longer accessible are invoked.

Finally, we discuss the optional CRUD sequencer component, see Figure 6. We allowed the definition of sequences of Business Schemas on the policy model that is used in the policy server, which can be turned on or off at any time. This meant that the client application has to follow a particular set of generated Business Schemas if it wants to perform some operation. To ease the development of applications, the Policy Extractor also creates a java class that uses the Business Manager to create instantiations of the first Business Schema of each sequence (*factory* object at lines 74 and 80). Then, each Business Schema has a method to instantiate the next one in the sequence (lines 76 and 81). We also allow the definition of sets of rules when a client moves in a sequence in the access control policy, e.g. moving from the first Business Schema in a given sequence to the second could prevent the client application from using the first one again.

This implementation of the CRUD sequencer has a couple of problems, however. First, to make sure that it can adapt to a large number of scenarios the sequence definition model must

```

74 S_Cust_1 = factory.get_S_Customers_all(session);
75 S_Cust_1.execute();
76 S_Orders = S_Cust_1.nextBE_S1(
77     Role_IRole_B1.s_orders_S_Orders_byShipCountry,
78     session);
79 S_Orders.execute(S_Cust_1, "Portugal");
80 S_Cust_2 = factory.get_S_Customers_all(session);
81 I_Orders = S_Cust_2.nextBE_S2(
82     Role_IRole_B1.i_orders_I_Orders_withCostumerID,
83     session);

```

Figure 6. S-DRACA CRUD sequence usage.

be flexible. Secondly, when the same Business Schema is used in more than one sequence it can potentially have a “next” method for each sequence it belongs to. Work is being done to address these problems.

Figure 6 shows the interface made available to the developers to use the CRUD sequencer. The factory class (lines 74 and 80) allows to obtain the first Business Schema in a sequence. When all the operations on that Business Schema are performed, the next Business Schemas can be requested using the “next” method (lines 76 and 81). These Business Schemas can be used normally as shown in Figure 5.

C. Performance Assessment

In order to evaluate the overhead induced by our access control stack solution, a performance assessment was carried out. Basically, we compared the initialization, instantiation and response time between the traditional solution using JDBC and the solution proposed in this paper (pushing the CRUD expressions to the server). We measured the time it took for the system to be ready to be used (obtain the connection object in JDBC or a Business Schema in S-DRACA), to execute a single Select expression when the server has 500 CRUD expressions stored, and for changes on the access control policy to be applied on the clients, along with the bandwidth used. We did not time the network security and authentication features, since they should always be implemented if the use case requires it, whether the proposed stack is used or not. For the adaptation process, we modified the access control policies 1000 times and measured the time it took for the changes to take effect on the clients and the bandwidth used.

The two solutions were implemented and tested in a PC with Windows 7 Enterprise and no network connection to prevent delays. The data was stored using SQL Server 2010 and all unnecessary processes were shut down. We verified the time it took for both solutions 10.000 times. Additionally, the select statement was executed on a table with 100 and on another with 100.000 records. We also cleared the DBMS cache between each execution.

For the initialization, the JDBC solution only took about 12 ms to provide a connection object, while the S-DRACA solution took 2905 ms. This time is explained with the initial configuration that is needed: requesting the access control policies, generate the access control mechanisms and instantiate them. Although this process takes a significant amount of time, it is only required once per session at the start. Regarding the execution of the select statement, the table with 100 rows showed an average increase of 4 ms when using the

solution proposed in this paper, from 1ms to 5ms. When targeting the table with 100.000 rows, the select statement took, on average, 1745ms to execute the CRUD statement directly while the solution proposed in this paper took 1750ms, showing a 5ms increase. The dynamic adaptation process took 10 ms on average to complete and around 350 bytes of bandwidth per Business Schema authorized, which also has the associated CRUDs information, and only 50 bytes if revoked. We can conclude, then, that the overhead introduced with our proposal is very small and can in most cases be neglected.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented an access control stack that aims at providing distributed and dynamic access control mechanisms that enforce the access control policies on the client side while maintaining the system secure. The main drawbacks of current solutions were identified: scalability, maintainability costs and security gaps. Scalability issues were overcome by deploying most of the access control mechanisms in client-side systems. Maintainability issues were overcome by providing automated processes to dynamically update the distributed access control mechanisms at runtime. The security gaps were overcome by: 1) implementing Sequence Controllers, 2) preventing client applications from directly using CRUD expressions and, finally, 3) by using secure connections between clients and DBMS. We also provided a proof of concept to empirically demonstrate that the presented solution is feasible.

A performance assessment has also been carried out to evaluate the impact of our proposal. The collected results show that its impact is unnoticeable when executing CRUD expressions or changing the access control policies. In opposite, the establishment of secure connections induced a significant overhead. Nevertheless, we cannot forget that this process is executed only once in each session. As a final conclusion, the stack herein presented shows to be a promising approach to overcome the identified drawbacks of most of the current approaches to enforce access control policies.

VII. REFERENCES

[1] P. Samarati and S. D. C. di Vimercati, "Access Control: Policies, Models, and Mechanisms," in *Foundations of Security Analysis and Design (LNCS)*, vol. 2171, Springer, 2001, pp. 137–196.

[2] S. D. C. di Vimercati, S. Foresti, and P. Samarati, "Recent Advances in Access Control - Handbook of Database Security," in *Handbook of Database Security*, M. Gertz and S. Jajodia, Eds. Springer, 2008, pp. 1–26.

[3] R. S. Sandhu and P. Samarati, "Access Control: Principle and Practice," *Commun. Mag. IEEE*, vol. 32, no. 9, pp. 40–48, 1994.

[4] Ó. M. Pereira, D. D. Regateiro, and R. L. Aguiar, "Extending RBAC Model to Control Sequences of CRUD Expressions," in *26th Intl. Conf. on Software Engineering and Knowledge Engineering*, 2014.

[5] Ó. M. Pereira, D. D. Regateiro, and R. L. Aguiar, "Role-Based Access Control Mechanisms Distributed, Statically Implemented and Driven by CRUD Expressions," in *ISCC'14 - 9th. IEEE Symposium on Computers and Communications*, 2014.

[6] Oracle, "Using Oracle Virtual Private Database to Control Data Access," 2011. [Online]. Available: http://docs.oracle.com/cd/B28359_01/network.111/b28531/vpd.htm#CIHBAJGI.

[7] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt, "Limiting disclosure in hippocratic databases," *30th Int. Conf.*

on Very Large Databases. VLDB Endowment, Toronto, Canada, pp. 108–119, 2004.

[8] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin, and J.-W. Byun, "On the correctness criteria of fine-grained access control in relational databases," *33rd Int. Conf. on Very Large Data Bases*. VLDB Endowment, Vienna, Austria, pp. 555–566, 2007.

[9] S. Barker, "Dynamic Meta-level Access Control in SQL," *22nd Annual IFIP WG 11.3 Working Conf on Data and Applications Security*. Springer-Verlag, London, UK, pp. 1–16, 2008.

[10] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending Query Rewriting Techniques for Fine-grained Access Control," *ACM SIGMOD Int. Conf. on Management of Data*. ACM, Paris, France, pp. 551–562, 2004.

[11] J. Eder, "View Definitions with Parameters," *2nd Intl Workshop on Advances in Databases and Information Systems*. Springer-Verlag, pp. 170–184, 1996.

[12] Y.-J. Hu and J.-J. Yang, "A semantic privacy-preserving model for data sharing and integration," *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*. ACM, Sogndal, Norway, pp. 1–12, 2011.

[13] L. E. Olson, C. A. Gunter, and P. Madhusudan, "A formal framework for reflective database access control policies," *15th ACM Int. Conf. on Computer and Communications Security*. ACM, Alexandria, Virginia, USA, pp. 289–298, 2008.

[14] L. E. Olson, C. A. Gunter, W. R. Cook, and M. Winslett, "Implementing Reflective Access Control in SQL," *23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. Springer-Verlag, Montreal, P.Q., Canada, pp. 17–32, 2009.

[15] A. Roichman and E. Gudes, "Fine-grained access control to web databases," *12th ACM symposium on Access Control Models and Technologies*. ACM, Sophia Antipolis, France, pp. 31–40, 2007.

[16] Canfora, G.; Visaggio, C.A.; Paradiso, V., "A Test Framework for Assessing Effectiveness of the Data Privacy Policy's Implementation into Relational Databases," in *Intl. Conf. on Availability, Reliability and Security*, 2009, pp. 240–247.

[17] J. Malenfant, M. Jacques, and F. Demers, "A tutorial on behavioral reflection and its implementation," *Proc. Reflect.*, 1996.

[18] B. Hicks, S. Rueda, D. King, T. Moyer, J. Schiffman, Y. Sreenivasan, P. McDaniel, and T. Jaeger, "An architecture for enforcing end-to-end access control over web applications," *15th ACM symposium on Access Control Models and Technologies*. ACM, Pittsburgh, Pennsylvania, USA, pp. 163–172, 2010.

[19] A. Chlipala, "Static checking of dynamically-varying security policies in database-backed applications," in *9th USENIX Conf. on Operating Systems Design and Implementation*, 2010, pp. 1–14.

[20] L. Caires, J. A. Pérez, J. C. Seco, H. T. Vieira, and L. Ferrão, "Type-based access control in data-centric systems," *20th European conference on Programming Languages and Systems: part of the joint European conferences on theory and practice of software*. Springer-Verlag, Saarbrücken, Germany, pp. 136–155, 2011.

[21] S. Chaudhuri, T. Dutta, and S. Sudarshan, "Fine Grained Authorization Through Predicated Grants," *IEEE 23rd ICDE - Int. Conf. on Data Engineering*. Istanbul, Turkey, pp. 1174–1183, 2007.

[22] J. Zarnett, M. Tripunitara, and P. Lam, "Role-based Access Control (RBAC) in Java via Proxy Objects Using Annotations," in *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, 2010, pp. 79–88.

[23] Y. Zhu, H. Hu, G.-J. Ahn, M. Yu, and H. Zhao, "JIF: Java + information flow," 2012. .

[24] H. Garcia-Molina, *Database Systems: The Complete Book*, 2nd E. 2008.

[25] S. Weil, S. Brandt, and E. Miller, "Ceph: A scalable, high-performance distributed file system," *Proc. 7th ...*, pp. 307–320, 2006.

[26] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003, pp. 29–43.

[27] M. Kerzner, "Hadoop Illuminated."

[28] IETF, "RFC 6101: The Secure Sockets Layer (SSL) Protocol Version 3.0."

Specifying and Dynamically Monitoring the Exception Handling Policy

Joilson Abrantes

Dep. of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte
Brazil
joilson@ppgsc.ufrn.br

Roberta Coelho

Dep. of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte
Brazil
roberta@dimap.ufrn.br

Abstract — The exception handling policy of a system comprises the set of design rules that specify its exception handling behavior (how exceptions should be handled and thrown). Such policy is usually undocumented and implicitly defined by the system architect. For this reason, developers may think that by just including catch-blocks in the code they can deal with exception conditions. This lack of information may turn the exception handling into a generalized “goto” mechanism making the program more complex and less reliable. This work proposes a domain-specific language called ECL (Exception Contract Language) to specify the exception handling policy and a runtime monitoring tool which dynamically checks this policy. The monitoring tool is implemented in the form of an aspect library, which can be added to any Java system without the need to change the application source code. We applied this approach to a large-scale web-based system and to a set of versions of the well-known JUnit framework. The results indicate that this approach can be used to express and to automatically check the exception handling policy of a system, and consequently support the development of more robust Java systems.

Keywords - exception handling; monitoring; dynamic analysis.

I. INTRODUCTION

Modern applications have to cope with an increasing number of abnormal computational states that arise as a consequence of faults in the application itself (e.g., access of null references), noisy user inputs or faults in underlying middleware or hardware. The exception handling (EH) mechanism [24] is one of the most frequently used techniques for developing robust systems, enabling such applications to detect and recover from these exceptional conditions.

Although exception handling mechanisms have been embedded in several mainstream programming languages (e.g. Java, C++, C#), studies have shown that the exception handling code is often poorly understood and the least-tested part of software systems [13], [14], [15], [16]. The exception handling behavior of a system is poorly understood, because it is generally spread over several implementation artifacts, and often the exception handling constructs (e.g., throw statements and try-catch blocks) lead the developers into believing that by just including EH constructs in the code that they can (i) deal with exceptional situations, and (ii) focus on the development of “happy path” scenarios [17]. This “ignore-for-now” approach may turn the exception handling into a generalized “goto” mechanism [17] making the program more complex and

even less reliable. As a consequence they may negatively affect the system, favoring the introduction of failures such as uncaught exceptions [30], [20] - which can lead to system crashes, making the system even less robust [5].

Work has shown that the lack of information about how to design and implement exceptional conditions leads to complex and spaghetti-like exception structures [12]. To the best of our knowledge, few studies have been proposed to better understand and check the exception handling behavior of systems. Some of them are based on the use of static analysis tools [20][21][22] and others on automated testing tools [1][7]. Both approaches, however, have intrinsic limitations.

The static analysis approaches [20][21][22] propose tools to discover the paths that exceptions take from signalers (i.e., elements that throw exceptions) to handlers (i.e., elements responsible for catching them). However, due to the limitations inherent in static analysis approaches combined with the characteristics of modern languages (e.g., inheritance, polymorphism and virtual calls) such approaches usually report many false positives. On the other hand, approaches based on the definition of test cases [1][7] limit the ability of checking exception handling behavior to the execution of test scenarios. Moreover, the high number of signaling and handling sites to be tested may lead to the test explosion problem [18][19].

None of the work mentioned above enables the developer to specify the exception handling behavior of a system and to check such behavior while the system is running on its production environment. Doing so, we could use the input data provided by real users or acceptance testers in order to check the exception handling behavior of a system.

This work proposes a domain-specific language (DSL) to specify the exception handling policy of a system (i.e., a set of design rules that specify the exception handling behavior of a system such as, how exceptions should be handled and thrown by its main elements). More often than not, such policies are not documented, and usually remain implicit in the form of a set of exception handling constructs spread over implementation artifacts. Moreover, this work also provides a runtime monitoring tool which verifies whether or not the exception handling behavior of a system is in accordance with the handling policy defined beforehand.

To evaluate the proposed language and the supporting tool, we conducted two case studies. We specified and checked the

exception handling policy of large-scale web system (SIRH) – which contains 593 KLOC of Java source code, 14781 throw statements and 2912 catch-blocks - and the well-known JUnit testing framework - for which four releases were evaluated. Results have shown that the proposed approach could be used to specify and dynamically check the exception handling policy of both systems. The contribution of this work is two-fold:

- It introduces a domain-specific language to specify the exception handling policy of a system.
- It presents a runtime monitoring tool implemented to support the dynamic check of the exception handling policy.

The remainder of this paper is organized as follows: Section II presents the main concepts related to this work; Section III presents a motivating example for using the proposed approach; Section IV presents the domain-specific language to define the exception handling policy and the supporting tool which checks such rules during runtime; Section V presents the case studies conducted in this work, and finally, Section VI presents the conclusions and future work.

II. THE EXCEPTION HANDLING MECHANISM

In order to support the reasoning for exception handling behavior of a system we present the main concepts of an exception-handling mechanism. An exception handling mechanism is comprised of four main concepts (i.e., the exception, the exception signaler, the exception handler, and the exception model - that defines how signalers and handlers are bound [14]) and two supporting concepts (i.e., the exception types and the exception interface) described next.

Exception Raising. An exception is raised by a method when an abnormal state is detected. In Java an exception is thrown using the throw statement [23].

Exception Handling. The exception handler is the code invoked in response to a raised exception. It can be attached to protected regions (e.g. methods, classes and blocks of code) [14]. In Java the handler is represented by the try-catch block [23].

Handler Binding. In many languages as in Java, the search for the handler to deal with a raised exception occurs along the dynamic invocation chain. This is claimed to increase software reusability, since the invoker of an operation can handle it in a wider context [11].

Exception Interfaces [11]: The caller of a method needs to know which exceptions may cross the boundary of the called method. In this way, the caller will be able to prepare the code beforehand for the exceptional conditions that may happen during system execution. For this reason, some languages provide constructs to associate a method's signature with a list of exceptions that this method may throw. However, they are most often neither complete nor precise [20], because languages such as Java provide mechanisms to bypass this mechanism by throwing a specific kind of exception, called *unchecked exception*, which does not require any declaration on the method signature.

Exception Types. Object-oriented languages usually support the classification of exceptions into exception-type hierarchies. The exception interface is therefore composed of the *exception types* that can be thrown by a method. Each handler is associated with an *exception type* that specifies its handling capabilities - which exceptions it can handle. In Java, exceptions are represented according to a class hierarchy, in which every exception is an instance of the Throwable class [23].

III. MOTIVATING EXAMPLE

Consider a layered-information system structured in three layers: the data layer (which accesses the database); the business layer and the presentation layer. One of the exception handling design rules that could be defined in this system is the following: ***the exceptions thrown by the Data layer should be a subtype of DAOException and should be handled in the Presentation layer.*** However, usually such rules are informally defined in the system documentation or, more often than not, remain undocumented as an implicit knowledge of the development team.

Both ways of dealing with the exception handling rules threaten the development of robust systems. Firstly, once documented such documentation may become outdated and be of little use. Secondly, the undocumented rules may become unknown for new members of the development team, and as a consequence, will not be followed. Moreover, none of these scenarios support the automatic check of such rules during system compilation or execution.

Let's consider that in such a system an instance of DAOException is thrown by the Data layer and is mistakenly handled by a generic handler defined in the Facade class (defined in the Business layer). *How can we check if the aforementioned rule is obeyed?* The approach shown in Section IV enables the developer to define and check such an exception handling rule.

IV. THE PROPOSED APPROACH

We propose an approach based on a DSL (Section IV-A) and a dynamic analysis tool (Section IV-B) to enable developers to define and verify the exception handling behavior of a system. More specifically, this approach allows the developer to create design rules for the exceptional flow, and check if such rules related to the exception handling code are neglected during the application execution.

A. The Exception Contract Language

We propose a domain-specific language called ECL (Exception Contract Language) whose main goal is to allow the creation of design rules for the exception handling behavior. Figure 1 partially illustrates the grammar of ECL language in BNF. In this version of BNF used, non-terminal symbols are written in bold, terminals are written with capital letters. In addition, the {} indicates zero or more repetitions of A. In order to simplify the reasoning of the grammar we omitted the definition of terminals such as ModID (which refers to a name of any identifier).

```

S: Rule
Rule: signaler QualifiedNameWithWildcardSignaler
exception SetOfNames
handler SetOfNames ;
SetOfNames: QualifiedNameWithWildcard {
  QualifiedNameWithWildcard }
QualifiedNameWithWildcardSignaler: QualifiedNameWithWildcard | *
QualifiedNameWithWildcard: QualifiedName |
QualifiedName+ | QualifiedName* | QualifiedName(..)
QualifiedName: ModID{.ModID}

```

Figure 1. Exception Contract Language (ECL) in Backus-Naur Form notation.

The main elements of ECL are:

- **signaler**: this element represents a method, class or package which can throw one or more types of exceptions.
- **exception**: identifies the types of exceptions thrown by the signaler.
- **handler**: this element represent the methods, classes or packages that will be responsible for handling the types of exceptions set to be launched by the signaler.

Figure 2 shows an example of an exception handling design rule created using ECL. This rule specifies that an exception of type `BusinessException` launched by `login(..)` method defined in `SignSystemBean` class, must be handled by any method defined on `LoginFilter` class.

```

signaler { br.com.ufrn.login.web.SignSystemBean.login(..) }
exception {br.com.ufrn.arq.business.BusinessException}
handler { br.com.ufrn.arq.filters.LoginFilter.* };

```

Figure 2. Example of ECL design rule.

The ECL language also supports the use of wildcards. The first is `*`: it matches any series of characters that can appear in a Java identifier. So, for example, in Figure 1 it matches all methods defined in `LoginFilter` class. The second is `+` wildcard, which can be combined over types. It means ‘match any subtype’. In Figure 1 we could add `+` to the exception name, and in doing so the contract would be related to `BusinessException` and its subtypes. We developed an Eclipse plug-in using `XText` framework to support the definition of design rules in ECL¹.

B. Dynamic Analysis of Exception Handling

A runtime monitoring tool was developed to check such rules while the program is running. This tool works in the background, analyzing whether defined design rules are being neglected. If a rule is not obeyed, a notification is sent to a remote server which will store the non-compliance. The remote server that receives such data, stores the notifications and provides this information to other applications which can generate EH reports, and mine such data.

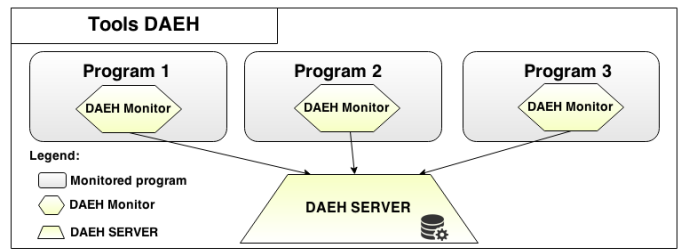


Figure 3. DAEH architecture.

The monitoring tool is called DAEH (Dynamic Analysis of Exception Handling), and consists of a set of monitors responsible for monitoring any Java application and a central server responsible for receiving the notifications from monitors and storing them. Figure 3 illustrates DAEH architecture. This architecture enables the implementation of other applications that communicate with the DAEH server which may query the monitoring information and perform any kind of data analysis.

C. DAEH Monitor

The DAEH monitor is added to the application to be monitored and performs the verification of exception handling design rules defined using ECL². Such a monitor is implemented as an aspect library which is combined at load-time with the application code to be monitored. This library was implemented using AspectJ and load-time weaving [8]. Since the monitor instrumentation is performed when the application classes are loaded into the Java Virtual Machine, there is no need to change the application source code. During the load-time weaving the DAEH monitor (i) loads the exception-handling design rules file and (ii) instruments every place where an exception is handled (every catch block). Hence, every time an exception is handled inside the system the DAEH monitor checks whether or not the handling action is breaking one of the existing exception handling design rules.

V. CASE STUDIES

This approach was used in two case studies: SIGRH - an enterprise large-scale web-based system developed in Java and the well known JUnit framework (from which 4 releases were used). Table I illustrates the characteristics of both systems.

TABLE I. METRICS OF SYSTEMS

Metrics	SIGRH	JUnit 4.6	JUnit 4.7	JUnit 4.8	JUnit 4.9
LOC	593.276	13098	14049	14373	15684
# of classes	3841	268	290	293	308
# of methods	51408	1724	1853	1885	2041
# of catch-blocks	2912	156	152	153	164
# of throw statements	1775	110	122	123	131

Since SIGRH had no exception design rules explicitly documented, we needed to talk with the system architect in

¹ The language manual and the Eclipse plug-in is available at: <https://bitbucket.org/jvidalabrantes/daeh-tool/wiki>.

² The ECL manual and plug-in and DAEH tool is available at: <https://bitbucket.org/jvidalabrantes/daeh-tool/wiki>.

order to document the exception handling policy in the form of ECL rules. As a result of this talk, five main exception handling design rules were documented. Table II illustrates one of them. This rule states that instances of `BusinessException` thrown by any method should be handled by any method of `ViewFilter` class.

TABLE II. EXAMPLE OF DESIGN RULE CREATED FOR THE MONITORED SYSTEM

Id	Exception Handling Design Rule
3	<pre>signaler { * } exception { br.ufm.arq.erros.BusinessException } handler { br.ufm.arq.web.ViewFilter.* };</pre>

After defining the rules, we added the DAEH monitor to the application server on which the SIGRH system was running for acceptance tests. In a 5-day period, the DAEH server received 12,027 notifications of broken design rules. Table III shows the number of violations per design rule.

TABLE III. NUMER OF DESIGN RULES VIOLATIONS (DRVs).

SIGRH		JUnit	
Contract Id	# DRVs	Version	# DRVs
1	6	4.6	0
2	0	4.7	0
3	12,015	4.8	0
4	6	4.9	0
5	0	-	-

As can be seen, only three design rules were violated (i.e., rules 1, 3 and 4). Figure 4 illustrates one of such notifications. Analyzing the violations associated with design rule 3 we observed that all of them were caused by 8 handlers defined in different locations outside the `ViewFilter` class (specified in the design rule illustrated in Table II). Such violations occurred often (i.e., 12,015) because the same pieces of code were exercised more than once during acceptance testing.

```
<Exception: class br.ufm.arq.erros.BusinessException >
expected: <Handlers: [br.ufm.arq.web.ViewFilter.*]>
but was <Handler: UserMBean.login()>
```

Figure 4. Design rule violation message.

The proposed approach was also used to define and monitor the exception handling design rules of the JUnit testing framework. The design rules were defined manually by inspecting the source code of the framework. Manual inspection was needed because the JUnit documentation had no reference to the exception handling policy adopted in the framework. This task was possible since JUnit is a small-scale framework and very well structured. As a result of this task we were able to define 9 exception handling design rules. Table IV illustrates two of them.

TABLE IV. EXAMPLE OF SET DESIGN RULES FOR JUNIT

Exception Handling Design Rule
<pre>signaler { org.junit.experimental.max.MaxHistory.readHistory(..) } exception { org.junit.max.CouldNotReadCoreException } handler { org.junit.experimental.max.MaxHistory.forFolder(..); };</pre>
<pre>signaler { * } exception { junit.framework.AssertionFailedError } handler { junit.framework.TestResult.* : junit.tests.* };</pre>

In order to exercise the framework and to check for exception design rule violations, we ran the test suite that comes with the framework and added the DAEH monitor to the JVM where the tests were executed. Although the rules were defined for version 4.6 of JUnit, we used the same set of rules to check the exception handling behavior of a set of subsequent versions (i.e., 4.6, 4.7, 4.8 and 4.9). Our goal was to check whether there had been changes in the exceptional behavior as the framework evolved. To our surprise, none of the specified contracts broke across the subsequent versions of JUnit. Such behavior can be explained by the fact that JUnit is a very stable framework and that although the exception handling design rules are not explicitly documented, they are adequately maintained by the development team.

VI. DISCUSSIONS

Exception handling policy: a global design problem. The definition of the exception handling policy is a global design problem [12]. However, none of the languages which have embedded EH mechanisms provide a way to specify and check such a policy. Due to this lack of guidance developers tend to focus their design activities on the normal behavior of the application [2], [3] and forget the exceptional behavior design [4]. In this work, we propose a language to express the exception handling policy of a system in the form of simple design rules, which link signaling and handling sites. Such sites can be methods, classes or packages. The ECL language and the supporting monitoring tool proposed in this work are the first step towards providing an infrastructure to help developers in specifying and analyzing the exception handling behavior of a system as a whole.

Limitations of the proposed approach. The way the exception handling policy is expressed in ECL could be improved to (i) specify the handling action (i.e., what should be performed inside the try-catch block) or to (ii) express a complete set of rules (i.e., if no rule is specified for a signaler no exceptions are allowed). However, the current grammar was sufficient to express all exception handling design rules needed during the execution of the case studies.

VII. RELATED WORK

Two approaches [1][7] extended the JUnit testing tool to support the definition of automated tests for the exception handling behavior. The limitations of both approaches are two-fold: the developer needs to manually implement each test case, and each test case focuses on one single exception flow (i.e. throw-catch pair) at a time. Since most Java systems may contain dozens or even hundreds of exception flows it is hard to choose which ones should be tested. Our approach tackles this limitation since the exception handling design rules involve

higher level modules than single methods (i.e., classes or packages), enabling the checking of several flows at a time.

Terra and Valente [9] proposed a dependency constraint language for specifying acceptable and unacceptable relations among the elements of a system architecture. Such restrictions are statically checked in order to detect the points in the source code that violate the defined relations. This language allows the developer to define which exceptions a given module (i.e., method, class or package) can throw. However, it does not address the handling capabilities of modules nor how handlers and signalers can be bound. Our approach supports the specification of both handling and signaling design rules and check such rules at runtime.

Brunet and Guerrero [10] proposed a tool called DesignWizard that enables the developer to define design rules in the same programming language of the analyzed application, in the form of a set of JUnit test cases. Although such a tool extends the JUnit framework, the checking of design rules is performed statically based on ASM framework. DesignWizard does not support the definition of design rules related to exception signaling and handling capabilities nor how they are bound.

Jin et al proposed JavaMOP [25] a monitoring framework specific to Java programs. [20]. JavaMOP allows the definition of properties based on event specifications and generates AspectJ code for monitoring - weaved into the target program in compile time. When a specification is validated or violated, user-defined actions are executed. User-defined actions can be any Java code from logging to runtime recovery. Our approach differs from JavaMOP as our approach is specific to the monitoring and checking of exception handling design rules. The syntax of ECL is simpler than the one needed to specify properties in JavaMOP, and there is a single action available in our approach (send the violation information to DAEH server).

VIII. CONCLUSIONS AND FUTURE WORK

This paper introduces a domain-specific language to specify the exception handling policy of a system, which is, more often than not, undocumented and implicitly defined – negatively impacting the system robustness [12]. This work also presents a runtime monitoring tool to support the dynamic checking of such an exception handling policy. Two case studies were conducted to evaluate the proposed approach. Our findings indicate that the approach can be used to specify and dynamically check the exception handling design policy of a system. We are currently working on evaluating the needs for adding new language constructs to ECL.

REFERENCES

- [1] R. Di Bernardo, R. Sales, F. Castor, R. Coelho, N. Cacho, S. Soares Agile Testing of Exceptional Behavior. In Proc. of 25th Brazilian Symposium on Software Engineering, 2011.
- [2] H. Shah, et al., Why do developers neglect exception handling In Proc. of the 4th International Workshop on Exception handling, 2008.
- [3] R. A. Maxion and R. T. Olszewski, Eliminating exception handling errors with dependability cases: a comparative, empirical study, Software Engineering, IEEE Transactions on, vol. 26, 2000.
- [4] H. Shah, Gerg, C. and M. J. Harrold., Why do developers neglect exception handling?. In Proc. of the 4th International Workshop on Exception handling, 2008.
- [5] F. Cristian. Exception handling and software fault tolerance. IEEE Trans. Comput. 31(6):531540, 1982.
- [6] J. Kienzle. On exceptions and the software development life cycle. In Proc. of the 4th International Workshop on Exception Handling, 2008.
- [7] R. Sales, R. Coelho, Preserving the ExceptionHandling Design Rules in Software Product Line Context: A Practical Approach, In Proc. of the 1st Workshop on Exception Handling on Contemporary Systems, 2011.
- [8] Raminivas Laddad. 2003. AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., Greenwich, CT, USA.
- [9] R. Terra and M. Valente. A dependency constraint language to manage object-oriented software architectures. Softw. Pract. Exper., 39(12):1073–1094, 2009.
- [10] J. Brunet, D. Guerrero, J. Figueiredo. Design Tests: An Approach to Programmatically Check your Code Against Design Rules. In Proc. of ICSE'09, 2009.
- [11] R. Miller and A. Tripathi, Issues with exception handling in object-oriented systems, In Proc. of ECOOP'97. 1997.
- [12] M. P. Robillard and G. C. Murphy, Designing robust Java programs with exceptions, In Proc. of FSE 2000.
- [13] A. Garcia, C. Rubira *et al.*, Extracting error handling to aspects: A cookbook, In Proc. of ICSM 2007. IEEE.
- [14] A. Garcia, C. M. Rubira, A. Romanovsky, and J. Xu, “A comparative study of exception handling mechanisms for building dependable object-oriented software,” *Journal of systems and software*, v. 59, n. 2, , 2001.
- [15] B. Cabral and P. Marques, “Exception handling: A field study in Java and .Net,” in *Proceedings of ECOOP 2007*. Springer, pp. 151–175.
- [16] R. Coelho, A. von Staa, U. Kulesza, A. Rashid, and C. Lucena, “Unveiling and taming liabilities of aspects in the presence of exceptions: a static analysis based approach,” *Information Sciences*, v.181, n.13, 2011.
- [17] D. Mandrioli and B. Meyer, *Advances in object-oriented software engineering*. Prentice-Hall, Inc., 1992.
- [18] M. Bruntink, A. V. Deursen, T. Tourwe. Discovering faults in idiom-based exception handling. In Proc. of ICSE'06, 2006.
- [19] G. J. Myers. The Art of Software Testing. New York: John Wiley & Sons, 2004.
- [20] R. Coelho, A. Rashid, A. Garcia, F. Ferrari, N. Cacho, U. Kulesza, A. von Staa, and C. Lucena, Assessing the impact of aspects on exception flows: An exploratory study, In Proc. of ECOOP 2008.
- [21] C. Fu, B. Ryder. Exception-Chain Analysis: Revealing Exception Handling Architecture in Java Server Applications. In Proc. of ICSE'07, 2007.
- [22] M. Robillard, G. Murphy. Static Analysis to Support the Evolution of Exception Structure in Object-Oriented Systems. In ACM Trans. Softw. Eng. Methodol, 2003.
- [23] J. Gosling, *The Java language specification*. Addison-Wesley Professional, 2000.
- [24] JB. Goodenough. Exception handling: Issues and a proposed notation. Communic. of the ACM 1975.
- [25] D. Jin, P. Meredith, C. Lee, G. Rosu. JavaMOP: Efficient parametric runtime monitoring framework. In Proc. of ICSE'2012.

DefDroid: Securing Android with Fine-Grained Security Policy

Chao Huang School of Software Shanghai Jiao Tong University Shanghai, China bujingyun_beta@sjtu.edu.cn	Shuohong Wang School of Computer Science Fudan University Shanghai, China sh_wang@fudan.edu.cn	Haiyang Sun Faculty of Informatics Università della Svizzera italiana Lugano, Switzerland haiyang.sun@usi.ch	Zhengwei Qi School of Software Shanghai Jiao Tong University Shanghai, China qizhwei@sjtu.edu.cn
--	--	--	--

Abstract—Android occupies the absolute dominant position in mobile operating system and has the largest market share. Meanwhile, Android faces the risk of malicious insiders leaking sensitive information. In this paper, we present DefDroid, a repackaging tool for enforcing security policies by modifying Android applications without root privilege. The main advantages of DefDroid are that it provides a user-friendly interface to configure fine-grained policies and it supplies multiple deployment methods. We have implemented policies aimed at three types of services of Android system, i.e., content provider, file system, and network. We choose 74 arbitrary applications from Android market and the experimental results show that the successful rate of repackaging applications is about 94.6% which effectively improve the privacy security of Android system while the increased overhead can be tolerated.

Keywords—Android; permission restriction; repackaging; bytecode instrumentation

I. INTRODUCTION

Android OS (developed by Google) spreads around the world and becomes one of the most important mobile operating systems. According to data from the International Data Corporation (IDC), Android occupies 84.4% of Smartphone OS Market Share in the second quarter of 2014 [1]. There are more than 1 billion Android devices activated around the globe and Android is the dominant mobile OS in the world [2].

According to the report of The Wall Street Journal [3], antivirus software only catches 45% of malware attacks and 55% malware behaviors are missed. In Android ecosystem, sensitive information leakage is a serious problem due to the rapid expansion of Android mobile phone customers. The native Android permission system follows an ‘all or nothing’ policy to restrict applications’ access to privileged resources. That is to say, an application cannot be successfully installed unless all permissions it applies for are allowed. This coarse-grained access control system brings terrible security problem and privacy leakage.

To overcome the issue, companies and researchers come up with several methods [4]–[6] to provide fine-grained security policy for Android which can be divided into three types. The first approach is native library rewriting which imposes fine-grained access control by intercepting the native calls. However, it does not allow security administrator to write

and deploy unified security policies according to different requirements of companies. The second one is fine-grained access control that can be achieved by modifying the Android operating system. The defect of it lies in that it demands root privilege and it is unable to get more high-level, unformed information. The last approach is based on bytecode instrumentation technique, which makes it possible to obtain detailed information of application behaviors at Java level and protect sensitive information at greater extent. However, there is no satisfactory method based on this technique (to our knowledge). Consequently, we propose a novel policy enforcement system using this technique.

Our Approach To protect sensitive information of Android system, we propose a novel and deployable tool with fine-grained security policy. DefDroid does not need root privilege, which avoids modifying the Android operating system. On the contrary, security policies provided by DefDroid are implemented into arbitrary applications by repackaging applications itself. The repackaging applications can be installed on users’ mobile devices after signing them with valid keys. The policies provided by DefDroid are implemented into applications through bytecode instrumentation. DefDroid translates user-defined security policies into instrumentation code and inserts these codes into specific code snippets to enforce security.

Theoretically, DefDroid is able to monitor all interaction with Android operating system except invocation of native code. DefDroid enables much more fine-grained policy enforcement than original permission model of Android and other solutions, such as FireDroid [4] and Aurasium [5]. There are three groups of policies and each group represents a service of Android operating system, i.e., content provider, file system, and network.

The main contributions of this paper are that

- We develop fine-grained policies to protect different types of sensitive information in Android system, such as protection of specific contacts and sandbox of file system.
- We provide groups of policies to customers so that users can customize policies to protect their sensitive information without knowing too much about implementation details.
- We show a workable solution to deal with bytecode instrumentation and repackaging applications.

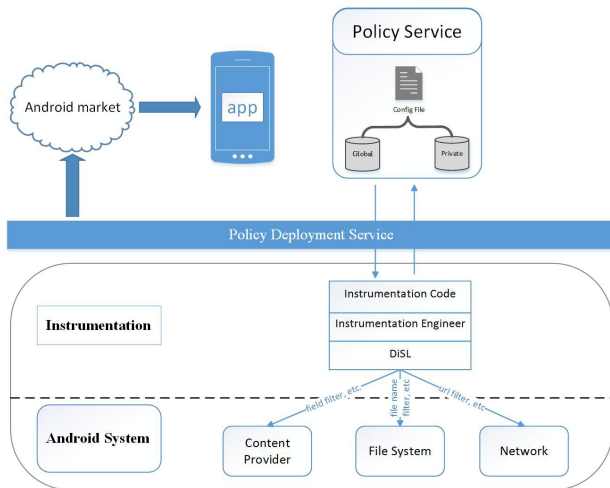


Fig. 1. Architecture of DefDroid

- We propose a scheme for companies who need to manage information security of multiple users' mobile device.

II. DEFROID SYSTEM DESIGN

The main idea of our proposal is that modifying source code of Android applications can completely enforce fine-grained permission policies and block malicious behaviors, although there are several problems need to be solved.

DefDroid system is made up of several components, as shown in Fig. 1. In order to implement security policies, a convenient way to define these policies is necessary. So, we create a configure file to edit and store security policies. The Policy Service (PS) is made up of configure files. All security policies written in the files are divided into two parts, namely Global policies and Private policies. The Global policies store security policies that influence all applications in private Android market. Private Android Market (PAM) is an Android application download platform inside the company or non-public Android market that serves specific customers. Applications in private Android market are repackaged and inserted relevant code. Policy Deployment Service (PDS) is the middle component of DefDroid. PDS gets configure file of PS and parses it to related code. Then, PDS transmits this policy information to Instrumentation component. Also, PDS takes the task of publishing Android package file to PAM. More specifically, PDS reads modified 'classes.dex' file, and other related files and compresses them to Android package file. Bottom layer is made up of Instrumentation part and Android System part. Instrumentation component contains instrumentation code, instrumentation engineer (IE), and DiSL. Instrumentation code is Java code based on grammatical rules of DiSL converted from PDS. IE is the core module to implement bytecode instrumentation. IE compiles instrumentation code to Java bytecode and inserts it into specific code snippets. The whole policies are divided into three parts, i.e., content provider, file system and network. Content provider is the standard interface that connects data

in one process with code running in another process [7]. A lot of sensitive information must be accessed through content provider, such as contact list, sms information, and media data. Policies aimed at content provider can effectively control information leakage in mobile phone. File system manages files and data stored in Android operating system. Policies of file system are able to protect data of storage devices, especially non-encrypted sensitive documents. Network is an important part of sensitive information protection. In several scenarios, some URL addresses are not allowed to be accessed because of security issues. Instrumentation of network service is able to enforce relevant policies.

III. IMPLEMENTATION DETAILS

In this section, we introduce the implementation details of DefDroid. Different from FireDroid [4] which monitors applications' interactions with the OS, DefDroid implements fine-grained permission policies of sensitive resource by bytecode instrumentation. Other related work, such as Aurasium [5] is proposed to solve it by replacing Android system libraries.

We use DiSL as our instrumentation tool. DiSL (domain-specific language for instrumentation) [8] is a domain-specific language especially designed for dynamic program analysis and is primarily designed to manipulate and transform bytecode. We decompress the Android package file and get 'classes.dex' file and invoke several functions of dex2jar [9] to convert the file to 'classes.jar'. After instrumentation is finished, we use Android dx tool [10] to convert jar file to dex file. Finally, we compress all related files into Android package file and sign applications with users' private keys before publishing to Android market.

For deploying fine-grained policies, there are two alternatives adapted in DefDroid. One is static policies. This means relevant instrumentation codes are going to carry out these policies. The other is dynamic policies. The approach will insert some codes in the application, these codes are able to get specific operations on sensitive resources from the server. Policy makers can rewrite data in the server to deploy security policies rapidly.

The configure file that contains policy setting is an xml format file, as shown in Fig. 2. We develop a component in PDS to parse the configure file. The 'Global' tag and 'Private' tag determine the scope of policies, and the 'name' attribute of 'Private' tag is the name of affected application. The next level contains three kinds of tags, i.e., 'ContentProvider', 'FileSystem', and 'Network'. Policies of DefDroid are mainly aimed to harden protection of sensitive resource of these three parts.

A. Content provider

Content provider is the standard interface to manage sensitive data, such as audio, images, and personal contact information. So, if we monitor 'ContentResolver' object, we can monitor all operations aimed at content providers. The Contacts provider, one of content providers, stores all information about contacts of mobile phones. DefDroid provides contacts

```

<?xml version="1.0" encoding="UTF-8"?>
<Global>
  <ContentProvider>
    <Contacts groupName="Company"
      operation="all">false<Contacts>
  </ContentProvider>
</Global>
<Private name="applicationName">
  <FileSystem>
    <File filePath="file path"
      operation="read">>true<File>
    <File extension="doc"
      operation="write">false<File>
  </FileSystem>
  <NetworkSystem>
    <network url="www.baidu.com" >false<Contacts>
  </NetworkSystem>
</Private>

```

Fig. 2. Example of configure file.

sandbox for specific fields. For example, most contacts have their own grouping, such as home, friend, and company, and DefDroid can block operations of contacts in specific group, as shown in Fig. 2. After being deployed, applications that have access to contact list are only enforced to operate on contacts in the ‘company’ group and other contacts are invisible.

B. File System

File system of Android operating system is similar to Linux system. Android has external storage and it’s necessary to add permissions for write/read access to external storage in AndroidManifest.xml. It is ubiquitous that applications have access to external storage, however, users who use their mobile phones frequently are more likely to store critical files in external storage of mobile devices. Hence, external storage is not safe.

DefDroid is able to manage files in external storage, as shown in Fig. 2. The policy means files with ‘doc’ extension are unwritable for applications. The value of ‘filePath’ will be appended to root path of external storage to get the absolute path of the file. And the policy will be aborted when the file is not found in the absolute path.

C. Network

Network is the most complicated interface in Android framework. Malware and virus widely exist in network. Building website blacklist to block website access is a workable way to protect mobile device and improve the security.

DefDroid is able to manage network access through URL address in external storage, as shown in Fig. 2. The ‘url’ attribute of ‘network’ tag stores website address and the policy means that accessing website by the URL are forbidden.

IV. PERFORMANCE EVALUATION

In this section, we present the performance evaluation result of DefDroid. We implement security policies presented in

TABLE I. EVALUATION RESULTS OF REPACKAGING

Type of App	Total	Repackaged	Success Rate
content provider	37	35	94.6%
file system	22	21	95.5%
network	15	14	93.3%

Section III. We evaluate repackaging performance and execution time before and after instrumentation. Our evaluation is conducted on a Google Nexus 7 Tablet running Android 4.4.2.

We download 74 applications from Android market, the functions of these applications correspond to policy model of DefDroid as introduced in Section III. There are 37 contacts applications, 22 file-related applications, and 15 network-related applications in total. Different applications are enforced different types of policies. For contacts applications, the policy is designed so that applications are only allowed to operate on contacts in ‘Company’ group. For file-related applications, we enforce a policy to prohibit write access of the applications to ‘data.txt’ file on the external storage. For network-related ones, applications are forbidden to visit website of “www.baidu.com” according to the policy.

As shown in Table I, there are altogether 4 applications failed to be repackaged. The reason is that the translator tool dex2jar still has several bugs resulting in several bytecodes translated from Dalvik bytecode not recognized by DiSL. Considering that the probability of this circumstance is very small, we ignore these questionable applications in our test set.

The experimental results of repackaging contacts applications, file-related applications and network related applications are shown as histograms in Fig. 3(a), 3(b), 3(c) respectively. The blue bins show the number of snippets modified by bytecode instrumentation (the left vertical axis displays their values) and the red ones show the execution time of repackaging each application (the right vertical axis displays their values). And the numbers on horizontal axis list each application of the corresponding type.

As shown in Table II, we measure the execution time of original and modified code snippets. The experimental results present the increased overhead caused by policies of DefDroid. For contacts, we need to query contacts table in database multiple times to get group id by group name and check if the contacts required by the applications are in this group. It definitely increases execution time compared with original code, and therefore we record these information of group id to avoid querying database repeatedly when requiring multiple contacts. According to the experimental results and user experience of modified applications, the overhead increases but can be tolerated. For policies of file system and network, we provide two modes of policy deployment, which are local and network respectively. Applications are forced to obtain data from Internet when implementation information is stored on servers, which will substantially increase execution costs. Therefore, it is recommended to store implementation information locally.

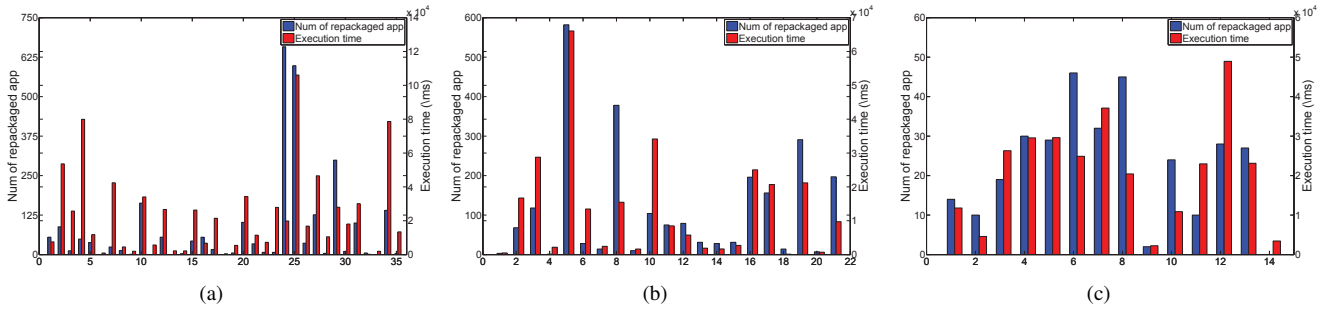


Fig. 3. Evaluation results of repackaging applications. (a). Evaluation results of repackaging contacts applications. (b). Evaluation results of repackaging file-related applications. (c). Evaluation results of repackaging network-related applications.

TABLE II. OVERHEAD MEASUREMENTS OF DEFROID'S POLICIES

	Original/(ms)	DefDroid/(ms)	Overhead
Group field sandbox of contacts	9	23	60.9%
File system restriction	2	2	0%
		157	98.7%
URL restriction of network	168	168	0%
		318	47.2%

V. RELATED WORK

In this section, we present an overview of the related work that aimed to deal with information leakage and malware detection.

Programing analysis has been applied to detect malicious applications in Android market. TaintDroid [11] is proposed and obtains good experiment results by employing dynamic taint analysis. Malware detection is able to clean malware applications in Android market. But malware that has already been installed in mobile devices are still doing malicious attacks.

Modifying Android application to enforce permission control policies through bytecode instrumentation is another approach to protect sensitive information. Aurasium [5] enforces policies by replacing Android's standard C libraries with Aurasium native libraries and bytecode instrumentation.

Ptrace is a tool used to track processes and modify system call. FireDroid [4] provides a fine-grained policy model by monitoring Android process with ptrace. Meanwhile, FireDroid contains a policy language to simplify policy making process. However, deploying FireDroid system is a difficult task, considering the high risk of rooting Android system.

VI. CONCLUSION

In this paper, we present DefDroid, a novel fine-grained permission control solution including policy server and application repackaging tool, which is able to act as defender of malware and malicious behaviors of applications for billions of Android mobile platform users. We implement DefDroid without rooting device, which reduces the potential risk of sensitive information leakage and is more acceptable to customers. The experimental results show that DefDroid increases overhead of applications compared with the original version.

However, considering the protection of sensitive information that DefDroid provides, the performance reduction can be tolerated.

VII. ACKNOWLEDGMENT

This work is supported by NSFC (No. 61272101), National R&D Infrastructure and Facility Development Program (No. 2013FY111900), and NRF Singapore CREATE Program E2S2. We thank support from Shanghai Key Laboratory of Scalable Computing and Systems for this research.

REFERENCES

- [1] *Smartphone OS Market Share, Q3 2014*. [Online]. Available: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>
- [2] *Android still the dominant mobile OS with 1 billion active users*. [Online]. Available: <http://www.engadget.com/2014/06/25/google-io-2014-by-the-numbers/>
- [3] *Symantec Develops New Attack on Cyberhacking*. [Online]. Available: <http://www.wsj.com/news/articles/SB10001424052702303417104579542140235850578>
- [4] G. Russello, A. B. Jimenez, H. Naderi, and W. van der Mark, "Firedroid: hardening security in almost-stock android," in *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 2013, pp. 319–328.
- [5] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications," in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 27–27.
- [6] A. Bartel, J. Klein, M. Monperrus, K. Allix, and Y. L. Traon, "Improving privacy on android smartphones through in-vivo bytecode instrumentation," *arXiv preprint arXiv:1208.4536*, 2012.
- [7] *Content Providers*. [Online]. Available: <http://developer.android.com/guide/topics/providers/content-providers.html>
- [8] L. Marek, A. Villazón, Y. Zheng, D. Ansaloni, W. Binder, and Z. Qi, "Disl: a domain-specific language for bytecode instrumentation," in *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*. ACM, 2012, pp. 239–250.
- [9] *Dex2jar*. [Online]. Available: <https://code.google.com/p/dex2jar/>
- [10] *Android apktool*. [Online]. Available: <http://code.google.com/p/android-apktool/>
- [11] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. 2010, pp. 1–6.
- [12] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 639–652.

Developers' importance from the leader perspective

Guilherme Costantin Tângari
Faculdade de Ciência da Computação
Universidade Federal de Uberlândia
Uberlândia, Brazil
guilhermecostantin@mestrado.ufu.br

Marcelo de Almeida Maia
Faculdade de Ciência da Computação
Universidade Federal de Uberlândia
Uberlândia, Brazil
marcelo.maia@ufu.br

Abstract—Several companies use the amount of deliveries as a metric of performance evaluation of the developer. However, the productivity of a developer and his importance for the company is not only related to the amount of lines of code produced. There are a variety of factors that can contribute to the relevance of a developer for a team. This paper aims at mapping some of these factors, measuring those that are more important for companies and propose an evaluation model of developer importance that considers more than just deliveries. We have found that some factors are more important than others and that there are minor differences for different companies. We have also developed a high accuracy classifier that can indicate the importance of the developer based on a set of attributes.

Keywords—productivity, developers' importance, pattern recognition, human factors

I. INTRODUCTION

All kinds of companies have been investing in techniques to increase productivity in order to increase competitiveness, and this is no different in the software industry, which still continues investing in new methods, tools and best practices that could lead organizations to productivity improvement [1].

However, unlike hardware, which improves their price-performance ratios by orders of magnitude per decade, software productivity seems to have trouble to evolve in a similar pace [2]. The current productivity rates are similar to the rates of decades ago (one to two lines of code per man-hour) [2]. Brooks et al. [3] states that there is no technical or management technique that by itself promises one order-of-magnitude improvement in software productivity, simplicity or reliability.

Traditional productivity metrics for software development are based either on lines of code (LOC) or function points (FP) [4], for example, the amount of LOC or FP developers deliver per hour. A slightly more abstract definition for productivity is the ratio of delivered outputs to consumed inputs, where outputs may be LOC, FP, or other relevant delivery, and inputs are the resources used to produce that output, e.g., time, people [2], [5], [6].

Nonetheless, the use of only these traditional metrics can mislead the management of software teams. LOC does not take into account the effort and knowledge required to write them. Complex problems often require experienced developers to solve them, and often, they do not require lots of LOC. In that case, experienced developers would be penalized.

There are other notions of productivity that are not also taken into consideration when evaluating just lines of code, for example, developers with greater experience or with knowledge in specific tools may be frequently consulted by other developers to streamline and improve the development process of the team as a whole, so the formers have an indirect notion of productivity.

Talent retention and team motivation, for example, are two fundamental issues for any software company [2], [7]–[11]. Software is made by people, and people, when have their work recognized and well evaluated tend to produce more and better. A performance evaluation that considers only one aspect, such as the number of deliveries, and does not take into account the different levels of difficulty and the purpose of the code, so the every day relationship with colleagues and the company would be compromised by unfair assessment, demotivating individuals and teams. Employee turnover its a common problem in software companies [12], and a high turnover rate would lead to productivity losses, in addition to the increased cost of hiring and training, and most importantly, the loss of talents that search for recognition in other companies.

Several studies are devoted to discover the factors that have influence in productivity of software development and maintenance activities [1], [4], [7], [8]. Understanding those factors and having some mechanism to evaluate productivity in a fair way could provide to software team leaders a better tool to evaluate and compare their developers. Several companies are beginning to gain awareness of these issues and are committed to improve the way they evaluate developer performance. This work aims to investigate how team leaders understand the notion of importance, indicating which factors are most relevant in their developer overall assessment. We are interested on the investigation of these questions:

1. What are the most important criteria used by leaders while assessing developers?
2. It is possible to build a developer's classifier with high accuracy, using the proposed criteria? This question can be refined in other two:
 - a. Is it possible to have a generic classifier, i.e., company-independent? or
 - b. Is it more appropriate to build customized classifiers for each company?

As mentioned before, the performance of developers is highly related with their productivity, within a classical concept of the amount of deliveries. Nevertheless, managers and leaders on their daily work with them have a different perception of each one.

In this paper, we show an elicitation of factors, based on previous studies, which can have an influence in the leader's evaluation. We conducted a survey with team leaders representing software companies, where they evaluated their developers based on those factors. The result was analyzed in attempt to recognize a pattern in their evaluation. That way, we identified factors that mostly influence the leaders evaluation about their developers, and also built a high accuracy classifier for developers' importance.

In the next section, we will present the factors that we will use to achieve the developers importance classification, and the studies from where those factors were retrieved. Section III will present the methodology used to create and conduct a survey with human subjects. Section IV presents the results, and Section V discusses those results. Finally, Section VI provides the conclusion.

II. IMPORTANCE FACTORS

In this section, factors used to represent concrete evaluation items for leaders about their developers are presented. Those factors will be used to define the metrics under the Goal-Question-Metric (GQM) approach used to design the survey elaboration.

Those factors were extracted from several studies available in the literature and were grouped into categories that present semantic affinity. Those categories will guide the formulation of the questions in the GQM model.

TABLE I. ELICITATION OF THE IMPORTANCE FACTORS

Groups	Importance factors	References
Technical characteristics	Past experiences	[5], [7], [8], [10], [13]–[20]
	Specialization (expert in some technology or tool)	
	Generalization (diversity of skills)	
	Solve complex problems	
Behavioral characteristics	Productivity (quantity of deliveries per month)	[5], [7], [8], [10], [17], [18], [19], [21],[22]
	The main behavior of the developer when faces a problem	
	Communication with the team members	
Individual characteristics	Willingness to help a colleague	[23],[24]
	Creativity	
	Entrepreneurship	
	Pro-activity	
Commitment to the team / company	Leadership	[22], [25],[26]
	Planning and organization	
	Focus on the costumers	
	Focus on the results	
	Time of work in the organization	

III. METHODOLOGY

To investigate the current practice of developers' evaluation, we decided to perform a survey with human subjects in real software companies to extract the desired information and analyze it. In this survey, we ask the respondent firstly to classify a subject developer and then fill the rest of the survey with the respective developer's characteristics. To analyze the obtained data, we decided to use automatic classification methods to get a clear view of how those characteristics affect leaders' classifications, and as a product we still may have a classifier that can be used to help leaders gain more insight about their teams' productivity.

This section is aimed at explaining how the survey was designed, and show how we conducted our data analysis obtained from that survey, including the criteria analysis and the classifier construction.

A. Survey

1) Goal-Question-Metric

We used an approach called Goal Question Metric (GQM) [27] that helped us define our survey. GQM is a top-down approach, that is based on the assumption that first, to measure something, you need to specify goals, from what is possible to derive questions that define those goals, and then specify the metrics that need to be collected to answer those questions.

To fulfill the purpose of a goal, we have to determine three coordinates:

- a) *Issue*: The subject/matter you are dealing with.
- b) *Object (process)*: What is the central object of the analysis.
- c) *ViewPoint*: Under whom perspective the analysis is being made.

TABLE II. shows our GQM model, with our purpose, the questions derived from it and the metrics defined to answer those questions.

For all those factors, the leader used a Likert scale with 5 options, ranging from "Very low" to "Very high", except from two factors: "The time of work in the organization", that receive a numeric value representing the months that the developers work in the organization, and "The main behavior of the developer when faces a problem", where the leader have to choose between one of the following options:

- Try to solve on your own (Introspective)
- Search in documentation or books (Introspective)
- Search or ask in Question and Answer sites and forums (Communicative)
- Ask helps for the team or leaders (Communicative)

TABLE II. GOAL QUESTION METRIC

Goal	Purpose	Measure
	Issue	the importance
	Object	of a developer
	Viewpoint	under the leader perspective

Question	What is the technical-skills level of that developer?
Metrics	Productivity
	Past experiences
	Specialization (expert in some technology or tool)
	Generalization (diversity of skills)
	Solve complex problems
Question	What is the social-skills level of that developer?
Metrics	The main behavior of the developer when facing a problem ^a
	Communication with the team members
	Willingness to help a colleague
Question	What is the level of these behavior characteristics in the developer's profile?
Metrics	Leadership
	Creativity
	Entrepreneurship
	Pro-activity
Question	How is the commitment of the developer with the company?
Metrics	Planning and organization
	Focus on the costumers
	Focus on the results
	Time of work in the organization ^a

^a Factors that had different types of evaluation

2) Survey application

We applied the survey remotely, to give the freedom that our respondent needs to answer the question. For that, we used the Google Form tool.

We also, to preserve the companies' privacy, we did not get any kind of identification, both for the respondent and the developer being analyzed. The only asked identification was the company name from where those evaluations are. This was necessary for a deeper investigation specific for cases where companies reach the minimum of 10 developers evaluated.

3) Participant characterization

The survey was applied to software companies that has a software development environment with a minimum hierarchical structure where exists the role of leaders, or managers, or chief engineers, etc. (for future references, we call that person, the leader). All participant companies work in their own products (they are not only software factories), but they vary in size, considering amount of employees (developers), sector of operation (ERP, Telecom, etc.) and may vary in used technologies.

The respondents of the survey are team leaders. We understand that they are the right people to do it because, unlike the owner or higher level managers, they are close enough to the daily work, and can judge who are the most important developers and why, even if they do not use a formal method to assess it. They should answer one assessment per developer, i.e., if they evaluated 10 developers to reach the minimum to have their company individually analyzed, they answered 10 questionnaires.

B. Feature Selection

In order to conduct the analysis to determine which factors are the most relevant and have major influence in the leader

evaluation, we use the WEKA[28] tool, an open-source software for data mining and machine learning.

Many real world problems, like ours, have a lot of features involved and only some of them are relevant to the target concept [29], in our case, the importance of a developer. To solve this issue, we will use a strategy called feature selection, where we select a subset of features to focus our attention, and ignore the rest to speed up learning, improve our classifier quality and achieve the best accuracy of the learning algorithm [29], [30].

The algorithm that we will use is called *GainRatioAttributeEval*, which is a single-attribute evaluator, that evaluates the attributes one by one independently and then rank them. Our feature selection will make a choice based on that ranking. That method does not eliminate the redundant attributes (only the irrelevant ones), but that is not a problem because we know all the attributes, and this kind of evaluator does not need a search method, what makes it very fast.

C. Classification

As a result of our survey, one dataset with several leaders' evaluations about the developers is generated. In this dataset machine-learning algorithms are applied to generate a classifier. The machine learning algorithms need two sets of data: one for training and one for testing, to verify the accuracy of the classifier. Fig. 1 shows the schema that best represent this scenario.

To evaluate the performance of a classifier, we used 10-fold cross-validation that divided the dataset in 10 equal parts (called folds), take 9 pieces to use for training and use the last piece for testing, and then do it 9 more times, always alternating the piece used for testing, that way, a single fold will be used 9 times for training and 1 for testing. The result will be the average of the 10 runs.

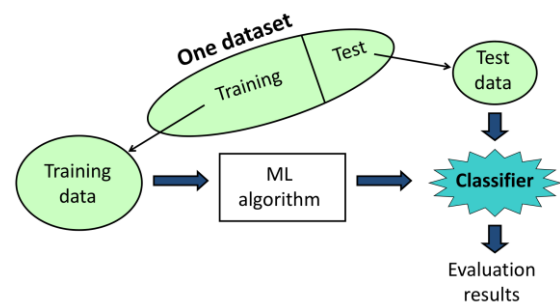


Fig. 1. Machine Learning algorithms schema

The used machine-learning algorithms are J48, a tree classifier, and Naïve Bayes, a bayesian classifier. There is no strong reason to choose them, but they tend to produce high quality classifiers in general, whenever possible.

J48 is a variation of a famous system called C4.5 which is described by Quinlan [28] that uses decision trees to build a classifier (WEKA actually let us have a look in the tree generated with all the weights).

Naïve Bayes is a probability method that has two assumptions: that the attributes are equally important and that

they are statistically independent (this independence assumption is never correct but the methods based on it often works well in practice).

We will also use a third algorithm called *AttributeSelectClassifier*, which actually use a method of feature selection (in our case, *Gain Ratio*) and an algorithm to perform the classification (in our case, J48 or Naïve Bayes). This way to apply feature selection only selects features in the training set, assuring we get more reliable results.

Finally, to conduct all those analysis, we will use a feature from WEKA called *EXPERIMENTER*, that allow us to run the same experiment more than one time and determine the mean and standard deviation, to avoid a misleadingly high or low accuracy based on the attribute selection to the training and testing sets. It also let us to compare the results of different algorithms.

IV. RESULTS

Following the steps presented in the previous section, we show the results of the application of the survey, the feature selection performed in the dataset generated by the survey and the application of the classifiers and their accuracy in the developers' classification.

A. Survey

Firstly, we present the data achieved with the survey application. Eleven respondents (leaders) provided 61 answers (unique developers evaluated). In a few cases, some leaders work at the same company, but they run different teams. There were eight companies involved in the collected data.

We asked for the leaders to classify the developer in five degrees of importance. Those degrees and the distribution of the 61 developers among them are shown in Fig. 2.

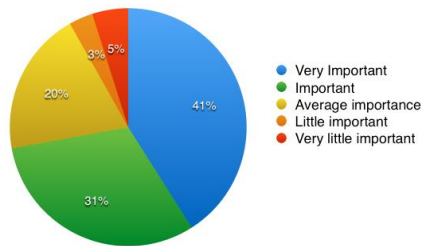


Fig. 2. Distribution of developers per degree of the class

TABLE III. NEW DEVELOPER'S SET OF CLASSES

New class	Original class
High importance	Very important
	Important
Low importance	Average importance
	Little important
	Very little important

Analyzing the results of the survey, we came to the conclusion that the leaders were conservative in some degree to

classify their developers in the lowest classification of importance.

From this analysis, we decided to group the developers also in only two classes based on the original five classes, as shown in TABLE III. in order to understand a more general picture of the intention of those leaders.

B. Feature Selection

As explained in the Section III.B, we used the algorithm *GainRatio* to rank the proposed attributes, in order to proceed with feature selection. TABLE IV. shows the rank ordered by the Average merit (the rate that the attribute influences in the classification) resulted of that algorithm application, using the original set of classes (five classes) and TABLE V. show the same view, now using the new set of classes (two classes).

TABLE IV. ATTRIBUTE RANKING (ORIGINAL SET OF 5 CLASSES)

Features	Average merit
Capacity of solving complex problems	0.303
Subjective evaluation of the productivity	0.29
Proactivity	0.226
Past experiences	0.211
Generalization (diversity of skills)	0.202
Specialization (expert in some technology or tool)	0.2
Time of work in the organization	0.184
Creativity	0.18
Focus on the results	0.167
Focus on the customer	0.148
Main behavior of the developer	0.14
Communication with the team members	0.137
Planning and organization	0.122
Leadership	0.097
Entrepreneurship	0.087
Willingness to help a colleague	0.084

TABLE V. ATTRIBUTE RANKING (NEW SET OF 2 CLASSES)

Features	Average merit
Proactivity	0.168
Subjective evaluation of the productivity	0.156
Capacity of solving complex problems	0.126
Focus on the results	0.112
Past experiences	0.107
Creativity	0.095
Planning and organization	0.09
Generalization (diversity of skills)	0.08
Specialization (expert in some technology or tool)	0.071
Focus on the customer	0.063
Time of work in the organization	0.063
Willingness to help a colleague	0.057
Leadership	0.052
Communication with the team members	0.039
Entrepreneurship	0.015
Main behavior of the developer	0.016

If we choose the three most relevant attributes, or expand our selection and choose the first ten, we will see that, even in a different order, they are the same, which supports our decision to group the class values.

C. Classification (all companies)

Considering that attributes have been ranked, we applied feature selection technique and use only the most relevant attributes in the classification. To determine how many features need to be selected to get a higher performance, we conducted an exhaustive test (we ran the classifiers with a crescent numbers of features selected, from 2 to 16) and chose the configuration with better performance (8 attributes). As we have two classes, we will show the results for the application of the J48 and Naïve Bayes for both of them, selecting the 8 first attributes more relevant and ignoring the rest.

TABLE VI. shows the results for the application of the algorithms using the first classification schema (5 classes). As we can see, J48 did not present good performance (close of 50% accuracy). Naïve Bayes had a better performance, but the accuracy could be considered still low for our purposes.

Now using the reduced set classes, we achieved better results, as expected, shown in TABLE VII. Again, Naïve Bayes had a better performance than J48, achieving now a relevant accuracy (85% of correctness).

TABLE VI. CLASSIFIERS APPLICATION (ORIGINAL SET OF CLASSES)

Algorithm	Percent correct
J48	51.88%
Naïve Bayes	61.12%

TABLE VII. CLASSIFIERS APPLICATION (REDUCED NEW SET OF CLASSES)

Algorithm	Percent correct
J48	75.88%
Naïve Bayes	85.21%

D. Classification (single company)

In our survey, out of the eight participant companies, 3 of them achieved the minimum numbers of responses that would allow an individual analysis of the company. We show the results of that individual analysis for one company (to preserve the company privacy, we call it *Company A*).

Company A evaluated 20 developers, and they presented reasonable distribution of the developers across the new classes (TABLE VIII.). TABLE IX. shows the attribute ranking for this particular company (we can see that there is a few differences from the relative generic attribute ranking in TABLE V. , which is discussed in Section V).

In this particular case, we had large difference between J48 and Naïve Bayes in what refers to feature selection. The selection did not produce a positive effect, and therefore classification performed better with all the attributes (results are shown in TABLE X.), and coincidentally they both presented the same accuracy. For this analysis, we considered only the new classification that proved to improve the accuracy of the classifiers.

TABLE VIII. DISTRIBUTION OF THE DEVELOPERS ACROSS THE REDUCED NEW SET OF CLASSES

High importance	10
Low importance	9

TABLE IX. ATTRIBUTE RANKING (INDIVIDUAL COMPANY)

Features	Average merit
Proactivity	0.323
Capacity of solving complex problems	0.298
Communication with the team members	0.254
Focus on the results	0.23
Creativity	0.206
Subjective evaluation of the productivity	0.187
Planning and organization	0.191
Specialization (expert in some technology or tool)	0.189
Past experiences	0.173
Entrepreneurship	0.171
Main behavior of the developer	0.172
Willingness to help a colleague	0.156
Focus on the customer	0.158
Leadership	0.137
Generalization (diversity of skills)	0.136
Time of work in the organization	0.123

TABLE X. CLASSIFIERS APPLICATION (INDIVIDUAL COMPANY)

Algorithm	Percent correct
J48 (with 2 features)	79.50%
Naïve Bayes (with all features)	79.50%

V. DISCUSSION

The first point considered in this discussion is the creation of the reduced new set classes. As shown in TABLE III. based on the original set of classes, that had five different classes, we grouped those 5 classes in only 2, creating a new set of classes that proved, as expected, to improve the performance of all the classification algorithms applied. As we could observe in TABLE IV. and TABLE V. , that this new reduced set of classes did not change the importance of attributes. So, this new classification scheme preserves the meaning of the original classification performed by the leaders because of the small variation in the attributes position. Moreover, we could observe that the classifier accuracy is around 80%, which gives a reasonable level of confidence on the coherence of the impact of the respective relevant factors on the importance level of developers.

The top 3 factors, which appear in both rankings, have a positive correlation with the class, which means that the better is the factor evaluation, the better is the position in the developer's importance classification. One of them is the productivity of the developer, under the leader perspective, where productivity represents the amount of work delivered. This was not a surprise because, as we mentioned in the beginning of the paper, because this is the classic metric to evaluate the developer's performance. On the other hand, the other two features bring new information to the discussion.

Capacity to solve complex problems lead to the opposite direction of the classic metric (amount of work delivered), because it often leads to the production of a lower rate of outputs (LOC or FP) over inputs (resources, time) consumed. This is an important qualitative point to consider whenever awarding high productive developers.

Proactivity is actually a required behavior characteristic of teams involved in the solution of complex problems instead of more canonical systems where the tasks are more predictable. The human resources area can conduct better hiring processes knowing that their software team leaders evaluated this as a fundamental requirement for developers.

The classification results evaluating all companies together with Naïve Bayes provided a classifier with 85.2% accuracy that can be considered a successful and useful result. The use of this classifier can help leaders conducting more coherent analysis of the team profile.

When analyzing an individual company, we noticed some major changes in some features' position in the feature ranking (TABLE IX.). Behavioral characteristics (*creativity*) and attributes related to the developer's commitment with the company (*focus on results*) in some cases were more important than the classic metric of productivity. We credit those differences to the culture and values of that particular company. So, different companies may assess the importance factors with some variation.

Finally, it is important to point out some few threats of the validity of this study. The limited number of developers and companies involved in this study may limit the generalization for other contexts. Nonetheless, we have observed several intersections in different companies that mitigate part of this threat. The classification provided by leaders tended to be more positive, maybe because they would not like to say that they maintain developers with low importance in their teams. The reduced classification mitigates part of this threat.

VI. CONCLUSION

In this study, we provided a set of criteria used by the leaders of IT companies to evaluate their developers, and also ranked those criteria, finding that capacity of solving complex problems, quantitative evaluation of productivity and proactivity were generally the most important factors.

Moreover, we created a high accuracy classifier, which can help, for example, the human resource managers to look for candidates that have the necessary needed characteristics and more potential to become an important part of the team.

A qualitative analysis, considering the culture of the company and their values, and the application of that classifier in the collaborators of open-source software repositories, to validate the results or spot the differences, could be suggestions of future work.

REFERENCES

[1] S. C. Sampaio, et al, "A Review of Productivity Factors and Strategies on Software Development," in *5th Proc. of ICSEA*, 2010, pp. 196–204.

[2] B. W. Boehm, "Improving Software Productivity," *Computer*, vol. 20, no. 9, pp. 43–57, 1987.

[3] F. P. Brooks Jr, "No silver bullet-essence and accidents of software engineering," *IEEE Comput.*, vol. 20, no. 4, pp. 10–19, 1987.

[4] S. Wagner and M. Ruhe, "A Structured Review of Productivity Factors in Software Development Technical," 2008.

[5] C. Walston and C. Felix, "A method of programming measurement and estimation," *IBM Syst. J.*, vol. 16, pp. 54–73, 1977.

[6] W. D. Yu, D. P. Smith, and S. T. Huang, "Software productivity measurements," in *Proc. of COMPSAC '91*, 1991, pp. 558–564.

[7] B. W. Boehm, et al, *Software Cost Estimation with Cocomo II*, Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[8] P. D. Chatzoglou and L. A. Macaulay, "The importance of human factors in planning the requirements capture stage of a project," *Intl Journal of Project Management*, vol. 15, pp. 39–53, 1997.

[10] R. A. Scudder and A. R. Kucic, "Productivity Measures for Information Systems," *Inf. Manag.*, v. 20, n. 5, pp. 343–354, 1991.

[11] H. Sharp, et al, "Models of Motivation in Software Engineering," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 219–233, 2009.

[12] L. Wallace, M. Keil, and A. Rai, "Understanding Software Project Risk: A Cluster Analysis," *Inf. Manag.*, vol. 42, no. 1, pp. 115–125, 2004.

[13] K. D. Maxwell and P. Forselius, "Benchmarking software development productivity," *Software, IEEE*, v. 17, pp. 80–88, 2000.

[14] R. D. Banker, S. M. Datar, and C. F. Kemerer, "A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects," *Manag. Science*, vol. 37, pp. 1–18, 1991.

[15] W. D. Brooks, "Software Technology Payoff: Some Statistical Evidence," *J. Syst. Softw.*, vol. 2, no. 1, pp. 3–9, 1981.

[16] G. R. Finnie, G. E. Wittig, and D. I. Petkov, "Prioritizing software development productivity factors using the analytic hierarchy process," *J. Syst. Softw.*, vol. 22, pp. 129–139, 1993.

[17] B. Lakhanpal, "Understanding the factors influencing the performance of software development groups," *Inf. and Softw. Tech.*, v. 35, pp. 468–473, 1993.

[18] J. Vosburgh, et al, "Productivity factors and programming environments," in *ICSE '84 Proceedings of the 7th International Conference on Software Engineering*, 1984, pp. 143–152.

[19] C. Wohlin and M. Ahlgren, "Soft factors and their impact on time to market," *Softw. Qual. J.*, vol. 4, pp. 189–205, 1995.

[20] C. Wohlin and A. Andrews, "Assessing Project Success Using Subjective Evaluation Factors," *Softw. Qual. J.*, v. 9, pp. 43–70, 2001.

[21] R. H. Rasch, "An Investigation of Factors That Impact Behavioral Outcomes of Software Engineers," in *Proceedings of the 1991 Conference on SIGCPR*, 1991, pp. 38–53.

[22] V. Lalsing, S. Kishnah, and S. Pudaruth, "People Factors in Agile Software Development and Project Management," *Int. J. Softw. Eng. Appl.*, vol. 3, pp. 117–138, 2012.

[23] R. E. Boyatzis, "Competencies in the 21st century," *J. Manag. Dev.*, vol. 27, no. 1, pp. 5–12, 2008.

[24] A. Shirazi and S. Mortazavi, "Effective management performance a competency-based perspective," *Int. Rev. Bus. Res. Pap.*, vol. 5, no. 1, pp. 1–10, 2009.

[25] C. Melo, et al, "Agile Team Perceptions of Productivity Factors," in *Agile Conference, 2011*, pp. 57–66.

[26] M. Coram and S. Bohner, "The impact of agile methods on software project management," in *Proc. of IEEE ECBS '05*, 2005, pp. 363–370.

[27] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," *Encycl. Softw. Eng.*, vol. 2, pp. 528–532, 1994.

[28] Mark Hall, et al (2009); *The WEKA Data Mining Software: An Update; SIGKDD Explorations*, Volume 11, Issue 1

[29] K. Kira and L. A. Rendell, "The Feature Selection Problem: Traditional Methods and a New Algorithm," in *Proc. of the Tenth National Conference on Artificial Intelligence*, 1992, pp. 129–134.

[30] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artif. Intell.*, vol. 97, no. 1–2, pp. 273–324, Dec. 1997.

Stability of Three Forms of Feature Selection Methods on Software Engineering Data

Huanjing Wang
Western Kentucky University
Bowling Green, Kentucky 42101
huanjing.wang@wku.edu

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida 33431
khoshgof@fau.edu

Amri Napolitano
Florida Atlantic University
Boca Raton, Florida 33431
amrfau@gmail.com

Abstract—One of the major challenges when working with software metrics datasets is that some metrics may be redundant or irrelevant to software defect prediction. This may be addressed using feature (metric) selection, which chooses an appropriate subset of features for use in downstream computation. There are three major forms of feature selection: filter-based feature rankers, which uses statistical measures to assign a score to each feature and present the user with a ranked list; filter-based subset evaluation, which uses statistical measures on feature subsets to find the best choice; and wrapper-based subset selection, which builds classification models using different subsets to find the one which maximizes performance. Software practitioners are interested in which feature selection methods are best at providing the most stable feature subset in the face of changes to the data (here, the addition or removal of instances). In this study we select feature subsets using fifteen feature selection methods and then use our newly proposed Average Pairwise Tanimoto Index (APTI) to evaluate the stability of feature selection methods. We evaluate the stability of feature selection methods on a pair of subsamples generated by fixed-overlap partitions algorithm. Four different levels of overlap are considered in this study. Four software metric datasets from a real-world software project are used in this study. Results demonstrate that ReliefF (RF) is the most stable feature selection method and wrapper based feature subset selection shows least stability. In addition, as the overlap of partitions increased, the stability of the feature selection strategies increased.

I. INTRODUCTION

For most software systems, superfluous software metrics (e.g., number of loops, number of global variables, and number of exit nodes) are often collected during the software development cycle. Some metrics may be redundant or irrelevant to software defect prediction. Therefore the identification and selection of a small set of relevant features from a metric dataset could be used by software developers to guide their efforts to reduce software development cost and produce more reliable software systems [1]. The identification and selection process is called metric (feature) selection. Feature selection algorithms (which select a subset of features from the original dataset) are often used to reduce the original feature set down to a subset containing only the most important features. Numerous feature selection methods have been proposed in the data mining and software engineering domains.

While feature selection is a necessary step, very little work has focused on the robustness (stability) of the feature selection methods in regards to software metrics data. The purpose of studying the stability of a feature selection technique is to determine which technique provides the feature subset that is the most robust to changes in the data. In this study, we used a fixed-overlap partitions algorithm which was proposed by our research group to generate a pair of subsamples which have same number of instances and a specified degree of overlap (fraction of instances in common). Then

feature subset chosen from the pair of subsamples using a feature selection method are compared. The proposed algorithm is different from the approaches used by most researchers, which either generate multiple random subsamples of the original dataset and compare the features chosen from these with one another, or compare the features from the subsamples directly with the features from the original data. It can be noted that the first approach can not control the overlap between subsamples, while the second approach compare features from different size of datasets.

The primary focus of this paper is to evaluate the stability of fifteen feature selection methods through a case study of four consecutive releases of a very large telecommunications software system (denoted as LLTS). We consider fifteen different feature selection strategies: three feature rankers each coupled with three feature subset sizes, the Correlation-based Feature Selection (CFS) filter-based subset evaluator, and wrapper-based feature selection using one of five different learners inside the wrapper. We evaluate the stability of feature selection methods on a pair of subsamples generated by the fixed-overlap partitions algorithm. Four different levels of overlap are considered in this study. We find that in general, the most stable feature selection methods is a ranker-based approach. Among the rankers, ReliefF (RF) is the most stable one. The trends of rankers being the most stable feature selection overall, CFS often being a moderate stable feature selection, and wrappers being extremely poor choices of feature selection in terms of stability were all present in the results. In addition, as the overlap of partitions increased, the stability of the feature selection strategies increased.

The rest of the paper is organized as follows. We review relevant literature on feature selection and stability in Section II. Section III provides detailed information about the three classes of feature selection, fixed-overlap partitions, and metrics used for measuring stability (including our newly-proposed extension on the Tanimoto Index) used in our study. Section IV provides a description of the software measurement datasets used and presents empirical results of our study. Finally, in Section V, the conclusion is presented and the suggestions for future work are indicated.

II. RELATED WORK

Feature selection is a necessary step in data mining. The main goal of feature selection is to select a subset of features that minimizes the prediction errors of classifiers. A number of papers have studied the use of feature selection techniques as a data preprocessing step. While many works have focused on the performance of models built using features selected by feature selection techniques, another way to evaluate a feature selection technique is through stability. Few studies exist on the stability of feature selection algorithms, but a

small number of studies have considered the stability of wrapper-based feature selection, which is often calculated in a similar fashion. Somol and Novovičová [2] conducted a comprehensive study of the stability of feature selection techniques and investigated the problem of evaluating the stability of feature selection techniques that produce subsets of varying size. They compared the stability of three wrapper techniques (Gaussian classifier, 3-Nearest Neighbor, and Support Vector Machines). Lustgarten et al. [3] proposed a stability measure called the Adjusted Stability Measure (ASM, based upon extending the consistency index to varying feature subset size), as opposed to Unadjusted Stability Measure (USM, based on the Jaccard index), that computes robustness of a feature selection technique with respect to random feature selection. They compared the stability of three wrapper approaches. Haury et al. [4] evaluated a number of feature ranking methods and one wrapper-based subset evaluation technique and considered stability in terms of how many features are in common between two subsets generated from independent subsamples of the original data. Dunne et al. [5] considered wrappers using a 3-nearest neighbor learner and three choices of search technique, evaluating stability by resampling the original dataset and finding the Hamming distance between the various feature subset masks. The overall stability is then defined by the average Normalized Hamming Distance. Kalousis et al. [6] used the Tanimoto coefficient that is a generalized version of the Jaccard index to measure similarity between two subsets of features. They concluded that stability provides an objective criterion to choose among feature selection algorithms. Selecting the most stable algorithm gives higher confidence in the quality of selected feature subset.

Few works consider the impact of dataset similarity when performing perturbation experiments, one paper, Alelyani et al. [7], does. In this paper, the authors note that without controlling for overlap, it is difficult to tell whether two feature subsets are different due to underlying stability issues with the ranker or due to differences in the datasets they were drawn from. To evaluate this, the researchers sampled 25% of the instances into one subset, and then created nine more subsets with exactly c of their instances in common with the first. The pairwise stability of the features from these subsets were evaluated as c varied from 0 to 1. They found that some algorithms were not able to outperform the inherent stability of the underlying datasets, and so should not be considered “stable” regardless of their stability performance. Although Alelyani et al. raises an important question about the role of dataset similarity, it does not necessarily address this question to the extent it deserves. Notably, during their experiments with varying the amount of overlap between subsets, only the overlap between the first subset and the remaining nine is considered; the overlap among the nine is not, and will depend on random chance. In addition, by consistently using only 25% of the instances from their datasets (which have as few as 85 instances to start with), they discard much of their data. Finally, although their proposal to compare a ranker’s stability with the minimum stability provided by the dataset is useful, it doesn’t address the problem of selecting stable rankers for different subset sizes, degree of class balance, size of underlying dataset, or difficulty of learning of the underlying dataset. These questions and more remain open.

Another work, Haury et al. [4], considers the role of overlap when considering the stability of gene subsets. In addition to other analysis of their datasets, the researchers consider the fraction of instances in common when comparing feature lists generated from subsamples of the original data which either have 80% or 0% overlap. They also compare feature lists among four distinct (but related) datasets. They found that the stability measures for the 0% overlap case more

closely resembled the between-datasets case than did the results from the 80% overlap case. However, unlike the 0% case, where it is noted that the original data was divided into two mutually-exclusive groups (which therefore have 0% overlap), for the 80% case the two groups were generated by adding 80% of the data from the original dataset into each group, and then splitting the remaining 20% in half and putting each half into one of the groups. Thus, the 80% refers to proportion of the original data shared by the two groups, not the overlap between the two groups. This makes it difficult to generalize the approach to create datasets with arbitrarily-chosen overlaps.

The main contribution of the present work is that we consider stability of three forms of feature selection techniques by comparing the selected features generated from two subsamples which have same number of instances and a specified level of overlap, rather than directly comparing separate subsamples of the original dataset with original datasets. In addition the Average Pairwise Tanimoto Index (APTI), which does not require feature subsets have the same size, is used to evaluate the stability of a feature selection technique.

III. METHODOLOGY

We consider fifteen different feature selection strategies: three feature rankers each coupled with three feature subset sizes, the Correlation-based Feature Selection (CFS) filter-based subset evaluator, and wrapper-based feature selection using one of five different learners inside the wrapper. The feature selection techniques are presented in Section III-A, while the Fixed-Overlap Partitions are discussed in Section III-B, and the stability measure is presented in Section III-C.

A. Feature Selection

Many techniques exist for choosing the optimal feature subset, but these can generally be placed into two categories: ranking-based methods and subset-based methods. Within the subset group, either filters or wrappers can be used to perform the actual evaluation. Filters are algorithms in which a feature subset is selected without involving any learning algorithm. Wrappers are algorithms that use feedback from a learning algorithm to determine which feature(s) to include in building a classification model. Feature rankers tend to be more efficient than subset-based methods, because a ranker need only provide a single score for each feature, and then subsets can be built based on ranked feature lists. For subset-based methods, different subsets must be considered, with the number of calculations reaching to 2^n (n is the number of features) if exhaustive search is used. Subset-based methods will take more computational resources than feature rankers.

1) *Feature Ranking*: For feature ranking, we choose three representative techniques: Relief (RF), Area Under the Receiver Operating Characteristic (ROC) Curve, and Signal-To-Noise (S2N). These were chosen for two reasons. First of all, they represent three major groupings of feature ranking technique: RF is a commonly-used algorithm for ranking features, while ROC is an example of threshold-based feature selection (TBFS) [8], and S2N is an example of first-order statistics-based feature selection (FOS) [9].

Relief is an instance-based feature ranking technique [10]. ReliefF is an extension of the Relief algorithm that can handle noise and multi-class datasets. When the ‘weightByDistance’ (weight nearest neighbors by their distance) parameter is set as default (false), the algorithm is referred to as RF.

Threshold-based Feature Selection Techniques (TBFS) were proposed and implemented by our research group [8]. In TBFS, each

attribute is evaluated against the class, independent of all other features in the dataset. After normalizing each attribute to have a range between 0 and 1, simple classifiers are built for each threshold value $\in [0, 1]$ according to two different classification rules (e.g., whether instances with values above the threshold are considered positive or negative class examples). The normalized values are treated as posterior probabilities and the performance of these probabilities is evaluated using a chosen metric, in much the same way that the posterior probabilities from a standard classifier would be evaluated. However, as the feature values are used directly, no actual classifier is built. In the present work, we used the Area Under the ROC Curve metric (ROC), which plots the True Positive Rate vs. the False Positive Rate over all possible threshold values and then uses the area under this curve as the performance of the posterior probabilities. When used as a ranker, this area is the quality of the feature.

First-order statistics-based feature selection (FOS) [9] is a family of related techniques which all center around the use of first-order statistics such as mean and standard deviation. Signal-to-noise (S2N) ratio is a technique in this family which is a measure used in electrical engineering to quantify how much a signal has been corrupted by noise. It is defined as the ratio of signal's power to the noise's power corrupting the signal. The S2N ratio can also be used as feature ranking method [11]. For a binary class problem (such as fp, nfp), the equation for signal to noise is:

$$S2N = (\mu_P - \mu_N) / (\sigma_P + \sigma_N) \quad (1)$$

where μ_P and μ_N are the mean values of that particular attribute in all of the instances which belong to a specific class, either P or N (the positive and negative classes). σ_P and σ_N are the standard deviations of that particular attribute as it relates to the two classes, respectively. If one attribute's expression in one class is quite different from its expression in the other, and there is little variation within the two classes, then the attribute is predictive. The larger the S2N ratio, the more relevant a feature is to the dataset [12].

For all three rankers, we considered three different feature subset sizes: 3, 4, and 5. These were chosen based on previous research [13] and to give a wider spectrum of the most common choices used for feature ranking on software metrics datasets.

2) *Filter-Based Subset Evaluation*: In this study, we evaluate one filter-based feature subset selection algorithms: Correlation-based (CFS) [14] feature subset selection. CFS employs the Pearson correlation coefficient [14], which can be calculated using the following formula:

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \quad (2)$$

In this formula, M_S is the merit of the current subset of features, k is the number of features, $\overline{r_{cf}}$ is the mean of the correlations between each feature and the class, and $\overline{r_{ff}}$ is the mean of the pairwise correlations between every two features. The numerator increases when the set of features is particularly good at classifying the data, while the denominator increases when the set has a significant amount of self-correlation, which implies redundancy.

3) *Wrapper-based Feature Subset Evaluation*: Wrapper-based feature subset selection is building a model using a potential feature subset and using the performance of this model as a score for the merit of that subset [15]. The wrapper-based feature selection methods employ some predetermined learning algorithms (classifiers or learners) to evaluate the goodness of the subset of features being selected. The performance of this approach relies on three factors: (1) the strategy to search the feature space for possible optimal feature

subsets; (2) the criterion to evaluate the classification model built with the selected subset of features; (3) and the learner.

Suppose a large set of n features is given, we need to find a small subset of features for future model building. Inspecting all candidate subsets (2^n) is impractical. There are some strategies that can solve the problem. One way is to use a search algorithm to generate the possible feature subsets. Based on preliminary experimentation, we chose the Greedy Stepwise approach, which uses forward selection to build the full feature subset starting from the empty set. At each point in the process, the algorithm creates a new family of potential feature subsets by adding every feature (one at a time) to the current best-known set. The merit of all these sets are evaluated, and whichever performs best is the new known-best set. The wrapper and CFS procedures terminate when none of the new sets outperform the previous known-best set.

During the search process, classification models are built using a potential feature subset and using the performance of this model as a score for the merit of that subset [15]. For our experiments the wrapper process uses five-fold cross-validation: the training set is divided into five equal folds (partitions), a classifier is trained on four folds, then tested on the last (fifth) fold. This process is repeated five times, and the results are averaged to give the merit of the potential feature subset. In this study, the classification models are evaluated using the Area Under ROC (Receiver Operating Characteristic) Curve (AUC) performance metric.

In this work, five diverse learners are used within the wrapper-based feature subset selector, consisting of naïve Bayes, multilayer perceptron, k -nearest neighbors, support vector machine, and logistic regression. The five learners were selected because of their common use in the software engineering and other application domains, and also because they do not have a built-in feature selection capability. Unless stated otherwise, we use default parameter settings for the different learners as specified in WEKA [16]. Parameter settings are changed only when a significant improvement in performance is obtained.

- 1) Naïve Bayes (NB) utilizes Bayes's rule of conditional probability and is termed 'naïve' because it assumes conditional independence of the features.
- 2) Multilayer Perceptron (MLP) is a neural network of simple neurons called perceptrons. Some related parameters of MLP were set as follows: the 'hiddenLayers' parameter was set to 3 to define a network with one hidden layer containing three nodes and the 'validationSetSize' was set to 10 (with 10% of the data being held aside for validating when to stop the backpropagation procedure).
- 3) K -Nearest Neighbors (KNN) [17], also called instance-based learning, uses distance-based comparisons. KNN was built with changes to two parameters. The 'distanceWeighting' parameter was set to 'Weight by 1/distance' and the 'kNN' parameter was set to 5.
- 4) Support Vector Machine (SVM), also called SMO in WEKA [16], had two changes to the default parameters: the 'complexity constant c' was set to 5.0 and 'build Logistic Models' was set to true. By default, a linear kernel was used.
- 5) Logistic Regression (LR) [18] is a statistical regression model for categorical prediction by fitting data to a logistic curve.

B. Fixed-overlap Partitions

Many approaches have been used to test the stability of feature selection techniques. Some take random subsamples from the original dataset and compare the features chosen on these subsamples with

each other; others compare the features chosen on the subsamples with those chosen from the original dataset. The first of these approaches has a known flaw: it does not control for the degree of overlap between the subsamples being compared (instead leaving this to random chance). This makes it difficult to determine whether the stability between feature subsets is due to similarity of the underlying datasets or is a property of the feature selection technique used. The second approach is somewhat limited in scope: although it is useful for observing stability in the case of adding or removing instances from a dataset, its use of two datasets of different sizes can impact how well the results generalize to other perturbation scenarios. Neither is able to evaluate how similar the feature subsets will be for two datasets which are equal in size and have a known degree of overlap. To address this, our research group proposed the Fixed-Overlap Partitions Algorithm [19] (Algorithm 1), which will create two new subsets that have the desired properties while also being as large as possible for the given degree of overlap. Note in this algorithm that c , the desired degree of overlap, can vary from 0 to 1, including the endpoints. A choice of $c = 0$ will find two entirely disjoint subsets, which will each contain half of the instances from the original dataset. On the other hand, $c = 1$ will create two copies of the original dataset which share all instances. This is generally not an interesting case to study, but is permitted by the algorithm.

Algorithm 1: Fixed-Overlap Partitions

input : Original dataset S with N instances
: c , the fraction of instances the two subsampled datasets should have in common ($0 \leq c \leq 1$)

output: Datasets S_1 and S_2 which have c of their instances in common while being identical in size and as large as possible for the given c

Let $d = 1/(2 - c)$ (e.g., $c = (2d - 1)/d$)
 S_1 and S_2 start out empty
Randomly select dN instances from S and add them to S_1
Randomly select cdN instances from S_1 and add them to S_2
Take all instances in S which are not in S_1 and add them to S_2

There are three properties which must be guaranteed when selecting these subsets: 1) that they contain the same number of instances, 2) that they have the specified degree of overlap, and 3) that they are as large as possible while the first two properties hold true (since there is no reason to discard instances if they could be used to improve feature selection or classification). Based on Algorithm 1, we can see that S_1 contains dN instances. To find the number of instances in S_2 , we note that two steps add instances to that dataset: one adds cdN instances and the other adds the instances not included in S_1 (e.g., $(1 - d)N$ instances). Working from here and using the definition of d in the algorithm, we have:

$$\begin{aligned}
|S_2| &= cdN + (1 - d)N \\
&= \left(\frac{2d - 1}{d}\right) dN + (1 - d)N \\
&= (2d - 1)N + (1 - d)N \\
&= 2dN - N + N - dN \\
&= dN
\end{aligned}$$

Thus, we have $|S_1| = |S_2| = dN$, satisfying the first property. As for the second property, recall that S_1 and S_2 share precisely cdN instances; thus, they have $cdN/dN = c$ of their instances in common, as desired. For the third property, observe that adding any instances

to either S_1 or S_2 would necessarily increase the fraction of overlap (since these would have to be instances already found in the other subsampled dataset). Thus, S_1 and S_2 are the largest datasets which are identical in size and have an overlap of precisely c . In this study, the degree of overlap is chosen from the set $\{0.25, 0.5, 0.7, 0.85\}$. A choice of $c = 0.85$ will generate two subsets with 0.87 ($d = 1/(2 - c) \times N$) instances.

C. Stability Measurement

In order to measure stability, first we have to decide the measurement metric. In this study we choose the Average Pairwise Tanimoto Index (APTI), derived from work originating in Kalousis et al. [6], since it does not require feature subsets have the same size. Let S_i and S_j be two different subsets of features. The original Tanimoto Index defines the stability between the two feature subsets as follows:

$$T(S_i, S_j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} = 1 - \frac{|S_i| + |S_j| - 2|S_i \cap S_j|}{|S_i| + |S_j| - |S_i \cap S_j|} \quad (3)$$

where $\frac{|S_i| + |S_j| - 2|S_i \cap S_j|}{|S_i| + |S_j| - |S_i \cap S_j|}$ is the Tanimoto distance between the two feature subsets. In this work, we propose Average Pairwise Tanimoto Index (APTI) that can be used to determine the stability of a set of feature subset pairs which are generated by same feature selection method on a pair of subsamples:

$$APTI(S^1, S^2) = \frac{1}{W} \sum_{i=1}^W T(S_{i1}^1, S_{i2}^2) \quad (4)$$

Here, we assume that S^1 and S^2 are paired feature subsets generated by same feature selection method on a pair of subsamples. For each pair of subsamples S_{i1} and S_{i2} , there are two corresponding feature subsets, S_{i1}^1 and S_{i2}^2 . The stability index (APTI) defined in Equation 4 varies in the interval of $[0, 1]$. As APTI is an average of Tanimoto Index values, its maximum of 1 represents the case where all pairwise comparisons are identical subsets, and the minimum of 0 means that no pairs ever have any features in commons. In our experiments, W is set to 30. That means for each dataset, 30 pairs subsamples are generated with certain level of overlap.

IV. EXPERIMENTS

A. Experimental Datasets

Experiments conducted in this study used software metrics and defect data collected from a real-world software project, and included data from four consecutive releases of a very large telecommunications software system (denoted as LLTS). The LLTS software system was comprised of several million lines of code. The data collection effort used the Enhanced Measurement for Early Risk Assessment of Latent Defect (EMERALD) system [20]. The software measurement dataset of LLTS contains data from four consecutive releases, which are labeled as SP1, SP2, SP3, and SP4. Each dataset includes 42 software metrics, including 24 product metrics, 14 process metrics, and four execution metrics. The dependent variable is the class of the program module: fault-prone (fp) or not fault-prone (nfp). A program module with one or more faults is considered fp , and nfp otherwise. Table I summarizes the numbers of the fp and nfp modules and their percentages in each dataset. A unique characteristic of these datasets is that they all are highly imbalanced datasets, where the proportion of fp modules is much lower than the nfp modules.

TABLE I
SOFTWARE DATASETS CHARACTERISTICS

	Data	#Metrics	#Modules	%fp	%nfp
LLTS	SP1	42	3649	6.28%	93.72%
	SP2	42	3981	4.75%	95.25%
	SP3	42	3541	1.33%	98.67%
	SP4	42	3978	2.31%	97.69%

TABLE II
STABILITY OF FEATURE SELECTION FOR SP1

Feature Selection	Overlap			
	0.25	0.5	0.7	0.85
RF, 3	0.8333	0.9000	0.8833	0.9333
RF, 4	0.8267	0.8400	0.8533	0.9067
RF, 5	0.6667	0.7111	0.7222	0.8111
ROC, 3	0.6400	0.7900	0.9167	0.9167
ROC, 4	0.5200	0.6000	0.6622	0.7778
ROC, 5	0.4266	0.4901	0.5909	0.7000
S2N, 3	0.4967	0.6000	0.5667	0.6500
S2N, 4	0.4159	0.5111	0.5644	0.6400
S2N, 5	0.4762	0.5540	0.6937	0.7667
CFS	0.4216	0.5223	0.5633	0.5972
Wrapper-NB	0.3973	0.5365	0.5696	0.6761
Wrapper-MLP	0.2645	0.3064	0.3419	0.3297
Wrapper-5NN	0.1679	0.1745	0.1988	0.2304
Wrapper-SVM	<i>0.1000</i>	<i>0.0931</i>	<i>0.1056</i>	<i>0.1413</i>
Wrapper-LR	0.3329	0.3504	0.4049	0.4707

B. Experimental Design

Experiments are conducted with fifteen different feature selection strategies on four software engineering metric datasets from a real-world software project. These feature selection strategies include three feature rankers each coupled with three feature subset sizes, the Correlation-based Feature Selection (CFS) filter-based subset evaluator, and wrapper-based feature subset selection using one of five different learners inside the wrapper. The goal of the experiments is to study how these feature selection methods can affect the stability of feature selection process. Thirty pairs of subsamples were generated from each original dataset with four different levels of overlap, and each feature selection method was applied to each pair of subsample. Once these feature subsets were created, the stability of the pairs of feature subset generated by same feature selection method were compared using our newly proposed Average Pairwise Tanimoto Index (APTI) described in Section III-C. In total, we calculate 240

TABLE III
STABILITY OF FEATURE SELECTION FOR SP2

Feature Selection	Overlap			
	0.25	0.5	0.7	0.85
RF, 3	0.6167	0.6167	0.7000	0.6500
RF, 4	0.6756	0.7156	0.6978	0.6978
RF, 5	0.7571	0.8254	0.9254	0.9778
ROC, 3	0.6800	0.8233	0.9167	0.9833
ROC, 4	0.6400	0.6933	0.7867	0.8800
ROC, 5	0.6143	0.6476	0.7206	0.7698
S2N, 3	0.4067	0.4233	0.5867	0.7067
S2N, 4	0.4438	0.5117	0.6889	0.7333
S2N, 5	0.4898	0.6333	0.7444	0.8175
CFS	0.45	0.49	0.54	0.64
Wrapper-NB	0.4259	0.5069	0.5417	0.6462
Wrapper-MLP	0.2015	0.2120	0.2783	0.2246
Wrapper-5NN	0.1461	0.1949	0.2898	0.2900
Wrapper-SVM	<i>0.0728</i>	<i>0.0787</i>	<i>0.0717</i>	<i>0.0739</i>
Wrapper-LR	0.2871	0.2780	0.4063	0.4401

TABLE IV
STABILITY OF FEATURE SELECTION FOR SP3

Feature Selection	Overlap			
	0.25	0.5	0.7	0.85
RF, 3	0.5667	0.6000	0.7000	0.7833
RF, 4	0.4889	0.6044	0.7511	0.7867
RF, 5	0.4675	0.5619	0.7381	0.8778
ROC, 3	0.2633	0.3633	0.4167	0.4000
ROC, 4	0.2590	0.3263	0.4006	0.4356
ROC, 5	0.2381	0.3636	0.4706	0.4964
S2N, 3	0.4833	0.6000	0.7800	0.8567
S2N, 4	0.4654	0.5473	0.7244	0.8133
S2N, 5	0.4516	0.5340	0.6353	0.6635
CFS	0.1685	0.2352	0.3022	0.3890
Wrapper-NB	0.2094	0.3133	0.3417	0.4135
Wrapper-MLP	0.1412	0.2274	0.2511	0.2667
Wrapper-5NN	0.1305	<i>0.1368</i>	0.2289	0.2706
Wrapper-SVM	<i>0.0862</i>	0.1589	<i>0.1223</i>	<i>0.1262</i>
Wrapper-LR	0.1767	0.2126	0.2759	0.2663

TABLE V
STABILITY OF FEATURE SELECTION FOR SP4

Feature Selection	Overlap			
	0.25	0.5	0.7	0.85
RF, 3	0.7167	0.7667	0.8000	0.8000
RF, 4	0.7911	0.8933	0.9333	1.0000
RF, 5	0.7349	0.8667	0.8556	0.8667
ROC, 3	0.0767	0.1433	0.2533	0.3800
ROC, 4	0.1254	0.2108	0.3321	0.4470
ROC, 5	0.1696	0.2696	0.4106	0.5362
S2N, 3	0.5600	0.7033	0.8067	0.8833
S2N, 4	0.5689	0.6711	0.7422	0.7378
S2N, 5	0.5639	0.6492	0.7429	0.8063
CFS	0.2634	0.3485	0.4231	0.5061
Wrapper-NB	0.2839	0.3426	0.5246	0.4964
Wrapper-MLP	0.1323	0.1673	0.1879	0.3063
Wrapper-5NN	0.1596	0.1510	0.2082	0.1978
Wrapper-SVM	<i>0.0379</i>	<i>0.0636</i>	<i>0.0784</i>	<i>0.0317</i>
Wrapper-LR	0.2004	0.2539	0.3282	0.3628

APTI values (4 datasets \times 15 feature selection \times 4 overlap levels).

C. Results and Analysis

Table II through Table V list the stability results for each dataset. These tables show the stability of subsets generated by each feature selection method (row) on subsamples with different level of overlap (column). For example, the first value in Table II, 0.8333, represents the stability of two feature subsets selected by Relief (RF) with feature subset size three and the overlap level of the pair of subsamples is 0.25. For each overlap level, the most and least value (stability) are printed in **bold** and *italics*, respectively. Figure 1 shows stability on average across all four datasets. From these tables and figure, we can observe the following facts:

- Overall, we can order the three classes of feature selection strategies from the most stability to least stability, ranker, filter-based subset evaluators, and wrapper-based subset evaluators. In terms of ranker, RF shows extremely high stability. The highest stability is found for RF with subset size four and overlap level 0.85 on dataset SP4. Followed by RF, ROC shows more stability than S2N for SP1 and SP2 datasets, while S2N shows more stability than ROC for SP3 and SP4 datasets. There are no patterns to show the relationship between feature subset size and stability of selected feature subsets.
- Comparing to other classes of feature strategies, the similarities of wrappers are low. Among the five wrappers, NB wrapper

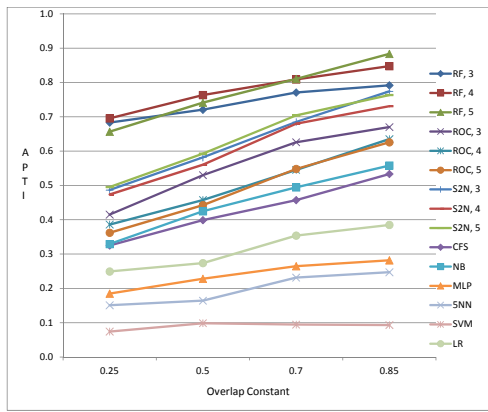


Fig. 1. Similarities of Feature Selection Methods

shows the most stability regardless of overlap level. The next similar wrapper is LR wrapper. SVM wrapper shows least stability, no feature subset pairs generated by SVM wrapper have a stability greater than 0.1 for SP2 and SP4 datasets and the stability value does not exceed 0.16 for SP1 and SP3 datasets. It is clear from these results that the choice of learner will have a very important effect on the chosen features.

- While intuitive, the results show that as the overlap of partitions increased, the stability of the feature selection strategies increased. This indicates that with enough change any selected subset become unstable.

V. CONCLUSION

Software metrics collected during project development play a critical role in software quality assurance. A typical project often collects large number of metrics. Metric (feature) selection plays an important role in data preprocessing step. By removing irrelevant and redundant features from a training dataset, software quality estimation based on some classification models may improve. One consequence of removing redundancy can be reducing stability: that is, the subset of chosen features may change significantly in the face of relatively small changes to the input dataset. In this paper, we propose a new metric for measure the stability on subset selected by feature selection techniques.

In this study, we present a stability analysis of of 15 feature selection methods (three feature ranking with three different subset sizes, one filter-based subset evaluator, and five wrappers) on a real-world software project. A newly-proposed variation of the Tanimoto Index (the Average Pairwise Tanimoto Index (APTI)) was used to evaluate the stability between subsets selected by feature selection methods. Experimental results demonstrate that the choice of feature selection methods has a major effect on the feature subsets. We find that there is the most stability (though not congruence) between the subsets chosen using rankers especially the RF ranker. The subsets selected by wrappers are even more dissimilar from one another. In addition, as the overlap of partitions increased, the stability of the feature selection strategies increased.

Future work may compare stability of a wide range of feature ranking techniques with more feature subset sizes, filter-based subset evaluators, and wrappers with different choices of learners and performance metrics. Experiments may be conducted on additional software metrics datasets from the software engineering domain.

REFERENCES

- [1] T. M. Khoshgoftaar, K. Gao, A. Napolitano, and R. Wald, "A comparative study of iterative and non-iterative feature selection techniques for software defect prediction," *Information Systems Frontiers*, vol. 16, no. 5, pp. 801–822, 2014.
- [2] P. Somol and J. Novovičová, "Evaluating stability and comparing output of feature selectors that optimize feature subset cardinality," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 11, pp. 1921–1939, 2010.
- [3] J. L. Lustgarten, V. Gopalakrishnan, and S. Visweswaran, "Measuring stability of feature selection in biomedical datasets," in *AMIA 2009 Annual Symposium Proceedings*, 2009, pp. 406–410.
- [4] A.-C. Haury, P. Gestraud, and J.-P. Vert, "The influence of feature selection methods on accuracy, stability and interpretability of molecular signatures," *PLoS ONE*, vol. 6, no. 12, p. e28210, 12 2011. [Online]. Available: <http://dx.doi.org/10.1371/journal.pone.0028210>
- [5] K. Dunne, P. Cunningham, and F. Azaaje, "Solutions to Instability Problems with Sequential Wrapper-Based Approaches To Feature Selection," *Machine Learning*, no. TCD-CD-2002-28, pp. 1–22, 2002.
- [6] A. Kalousis, J. Prados, and M. Hilario, "Stability of feature selection algorithms: a study on high-dimensional spaces," *Knowledge and Information Systems*, vol. 12, no. 1, pp. 95–116, May 2007.
- [7] S. Alelyani, Z. Zhao, and H. Liu, "A dilemma in assessing stability of feature selection algorithms," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, Sept. 2011, pp. 701–707.
- [8] H. Wang, T. M. Khoshgoftaar, and J. Van Hulse, "A comparative study of threshold-based feature selection techniques," in *2010 IEEE International Conference on Granular Computing, GrC 2010, San Jose, California, USA, 14-16 August 2010*, 2010, pp. 499–504.
- [9] H. Wang, T. M. Khoshgoftaar, R. Wald, and A. Napolitano, "A study on first order statistics-based feature selection techniques on software metric data," in *The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27-29, 2013.*, 2013, pp. 467–472.
- [10] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of 9th International Workshop on Machine Learning*, 1992, pp. 249–256.
- [11] C.-H. Yang, C.-C. Huang, K.-C. Wu, and H.-Y. Chang, "A novel gtaguchi-based feature selection method," in *IDEAL '08: Proceedings of the 9th International Conference on Intelligent Data Engineering and Automated Learning*, Berlin, Heidelberg, 2008, pp. 112–119.
- [12] M. Wasikowski and X. wen Chen, "Combating the small sample class imbalance problem using feature selection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1388–1400, 2010.
- [13] H. Wang, T. M. Khoshgoftaar, and N. Seliya, "How many software metrics should be selected for defect prediction?" in *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, May 18-20, 2011, Palm Beach, Florida, USA*, 2011.
- [14] M. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, The University of Waikato, Hamilton, New Zealand, April 1997.
- [15] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, Dec. 1997. [Online]. Available: [http://dx.doi.org/10.1016/S0004-3702\(97\)00043-X](http://dx.doi.org/10.1016/S0004-3702(97)00043-X)
- [16] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.
- [17] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 1573–0565, January 1991.
- [18] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Applied Statistics*, vol. 41, no. 1, pp. 191–201, 1992.
- [19] H. Wang, T. M. Khoshgoftaar, R. Wald, and A. Napolitano, "A novel dataset-similarity-aware approach for evaluating stability of software metric selection techniques," in *IEEE 13th International Conference on Information Reuse & Integration, IRI 2012, Las Vegas, NV, USA, August 8-10, 2012*, 2012, pp. 1–8.
- [20] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand, "EMERALD: Software metrics and models on the desktop," *IEEE Software*, vol. 13, no. 5, pp. 56–60, September 1996.

Building a Large-scale Software Programming Taxonomy from Stackoverflow

Jiangang Zhu

School of Software
Shanghai Jiao Tong University
jszjgtws@sjtu.edu.cn

Beijun Shen*

School of Software
Shanghai Jiao Tong University
bjshen@sjtu.edu.cn

Xuyang Cai

School of Software
Shanghai Jiao Tong University
bakercxy@hotmail.com

Haofen Wang*

East China University of
Science & Technology
whfcarter@ecust.edu.cn

Abstract—Taxonomy is becoming indispensable to a growing number of applications in software engineering such as *software repository mining* and *defect prediction*. However, the existing related taxonomies are always manually constructed. The sizes of these taxonomies are small and their depths are limited. In order to show the full potential of taxonomies in software engineering applications, in this paper, we present the first large-scale software programming taxonomy which is more comprehensive than any existing ones. It contains 38,205 concepts and 68,098 subsumption relations. Instead of learning from an open domain, we focus on taxonomy construction from Stackoverflow which is one of the largest QA websites about software programming. We propose a machine learning based method with novel features to create a taxonomy that captures the hierarchical semantic structure of tags in Stackoverflow. This method executes iteratively to find as many relations as possible. Experimental results show that our approach achieves much better accuracy than baselines. Compared with taxonomies related to software programming which are extracted from the general-purpose taxonomies such as WikiTaxonomy, Yago Taxonomy and Schema.org, our taxonomy has the widest coverage of concepts, contains the largest number of subsumption relations, and runs up to the deepest semantic hierarchy.

Keywords—Taxonomy Construction, Stackoverflow, Software Engineering

I. INTRODUCTION

Taxonomy plays an important role in software engineering. For example, in software maintenance such as measuring quality and predicting defects, taxonomies are used to measure the relatedness between documents and create links between bugs and committed changes [1]. In program comprehension, taxonomies provide an effective way to compute the semantic similarities between words from the comments and identifiers in software [2].

However, most existing taxonomies used in these applications are often manually created according to application specific requirements and their sizes are not large enough. A recent literature [3] argued that the quality and the scale of taxonomy would significantly benefit the performance when applied in software engineering. On the other hand, there have been a considerable amount of research works on taxonomy construction [4], [5], [6], [7], [8]. The value of automatic taxonomy construction is two folds. Automatic taxonomy construction can achieve large scale taxonomies while manual construction is a laborious process. Moreover, compared with automatic approaches that are data-driven, taxonomies built manually are

*Corresponding author



Fig. 1: An example from Stackoverflow

often highly subjective. Unfortunately, the resulting taxonomies of these existing automatic approaches would probably lead to poor results when applied in software engineering for several reasons. First is *timeliness*. The techniques in software engineering are fast changing, while the general web pages and encyclopedic sites are insensitive to this change and always fail to update in time. So it is not suitable to select text corpora such as general web pages or Wikipedia as its input. Second is *granularity*. Since the input of traditional taxonomy construction approaches is from an open domain, some fine-grained terms about software programming cannot be found in these taxonomies. For example, “hashmap” about a well-known data structure is not included in either Yago Taxonomy [9] or WikiTaxonomy [4] which are the largest existing public available taxonomies.

Recently, Stackoverflow has becoming one of the largest QA websites about software programming. Specifically, *questions* are the key elements in Stackoverflow. Besides the question description, as shown in Fig. 1, a question is also associated with tags and authors. Formally, a question q is a triple in form of (t_q, b_q, TS_q) , where t_q is the title of the question, b_q is the body while TS_q is the tag set which annotate the question. A *tag* a in Stackoverflow can be represented as a

four-tuple (t_a, d_a, c_a, Q_a) , where t_a is the name of the tag, d_a is the description of a , c_a is the number of questions that it has annotated, Q_a is the set of questions annotated by a . These tags represent vocabularies about software programming. They can also reflect the fast changing nature of technique terms because they are created on the fly by Web users. So the large amount of tags provide a promising way to build the taxonomy. Therefore, in this paper, we try to construct a software programming taxonomy from tags in Stackoverflow.

The problem is non-trivial and poses unique technical challenges. First, tags in Stackoverflow are always composed of domain-specific terms. While natural language processing techniques like segmentation or pos tagging are basis of some Web-based approaches for taxonomy construction, directly applying these approaches to our scenario will lead to poor results. Second, most tag-based taxonomy construction approaches only use tag co-occurrences and annotated documents to help the subsumption detection between tags [8], [7]. However, Stackoverflow contains unique information such as wiki descriptions of tags. We argue that these information will significantly increase the performance of taxonomy construction. How to design an algorithm to incorporate these information when detecting subsumptions between tags has not been studied yet. Moreover, there are tens of thousands of tags in Stackoverflow, it is time-consuming and sometimes impractical to enumerate all tag pairs for subsumption relation detection. How to perform subsumption relation detection in a large scale scenario is also a challenge problem.

In order to solve the above challenges, we propose a machine learning based approach. Specifically, our approach leverages several features from different aspects to measure the semantic relatedness between tags. Then a semi-supervised learning method is used to predict subsumption relations. To the best of our knowledge, we are the first to focus on building a software programming taxonomy from Stackoverflow. Our contributions mainly include:

- To overcome the informality of tags, we leverage different information from Stackoverflow to represent these tags. Specifically, we design the co-occurrence-based features to measure the semantic relatedness. Moreover, we also use the wiki description and annotated questions of each tag to learn a topic representation by applying Latent Dirichlet Allocation (LDA) [10]. Together with a lexical feature, our proposed approach can combine evidences from the co-occurrence relevance, the implicit topic relevance and the lexical relevance.
- We propose a unified model that incorporates these features to automatically construct a software programming taxonomy. Specifically, we design a blocking mechanism to reduce the number of tag pairs to be calculated which ensure our approach can be applied to a large scale scenario. Then, we treat subsumption relation detection as a binary-class classification problem to solve. A semi-supervised learner is applied which executes iteratively to find as many relations as possible. Note that the blocking mechanism and semi-supervised learning make our approach quite general and can be applied to other domains.

- We present a taxonomy about software programming. Experimental results show that our taxonomy not only contains a large number of concepts and subsumption relations, but also have a deep semantic hierarchy. That is to say, concepts and subsumption relations in our taxonomy are more fine-grained and can be applied in many software engineering applications.

II. RELATED WORK

There have been a considerable amount of research works on taxonomy construction. Approaches for automatic taxonomy construction can be encyclopedic-based or Web-based.. For the encyclopedic-based approaches, they mainly focus on extracting concept hierarchies from Wikipedia. WikiTaxonomy [4] builds a taxonomy from the Wikipedia category system. It contains 105,000 subsumption relations with the accuracy of 88%. Kylin Ontology Generator (KOG) [5] uses Markov Logic Network (MLN) to predict subsumption relations between Wikipedia infobox classes. Yago [9] interlinks Wikipedia categories to WordNet synsets. There are over 200,000 classes and 400,000 subsumption relations in Yago and the accuracy is estimated to be 96%. Our research is quite different from the encyclopedic-based approaches because there has been no structure information between tags in Stackoverflow.

Regarding Web-based approaches, it can be free text based or social tag based. For the free text based approaches, Hearst patterns [11] are widely used. The most recent effort is Probase [12]. It builds the largest taxonomy which contains over 2.7 million classes from 1.7 billion web pages. For the social tag-based approaches, Mianwei Zhou et al. [6] introduced an unsupervised model to automatically derive hierarchical semantics from social annotations. Jie Tang et al. [7] proposed a learning approach to capture the hierarchical semantic structure of tags. Xiance Si et al. [8] proposed three methods to estimate the conditional probability between tags and used a greedy algorithm to eliminate the redundant relations. Huai ren Lin et al. [13] described an integrated method for extracting ontological structure from tags that exploits the power of low support association rule mining supplemented by an upper ontology such as WordNet. Zhishi.schema [14] is the first effort to publish a general taxonomy from tags and categories in popular Chinese Web sites. These traditional tag-based approaches for a general domain only use the annotated documents to help the subsumption detection. However, Stackoverflow is more domain specific which contains other information such as wiki descriptions. So traditional tag-based approaches may not be the best because the additional information in Stackoverflow will probably increase the performance of taxonomy construction.

Regarding taxonomy construction in software programming, the most recent research is *Lexical Views* [3]. It applied some natural language processing techniques to automatically extract and organize concepts from software identifiers in a WordNet-like structure. But more software programming terms would not be included in this taxonomy since *Lexical Views* only use the software identifiers as its input. In our research, we first focus on automatically constructing a software programming taxonomy from Stackoverflow. Specifically, we design several novel features which can capture similarities between tags from several aspects. Also, we use a more

sophisticated semi-supervised learning approach by generating labeled examples semi-automatically.

III. APPROACH

In this section, we start with a brief introduction of our proposed approach, and then describe it in details.

A. Approach Overview

We now provide a workflow to explain the whole process and how different components interact with each other. As shown in Fig. 2, we have four main components, namely *Candidate Selection*, *Labeled Data Generation*, *Feature Engineering*, and *Semi-supervised Learning*. The input of *Candidate Selection* is tags collected from Stackoverflow. *Candidate Selection* tries to divide all tags into blocks. Each block includes similar tags which can form candidate tag pairs for further processing. *Candidate Selection* leads to a significant reduction of the number of tag pairs to be further processed for subsumption relation detection, which guarantees the scalability and efficiency of our approach. We generated labeled data (both positive and negative) semi-automatically using a rule-based method. All the tag pairs are fed to *Feature Engineering* to extract features like co-occurrence-based features and the topic-based features. A semi-supervised learning algorithm is adapted to discover hypernym-hyponym relations. The learned classifier can be updated iteratively by adding new labeled data of high confidence. Finally we build a software programming taxonomy composing subsumption relations between tags.

B. Candidate Selection

Since Stackoverflow contains tens of thousands of tags and the number is still increasing, it is time-consuming and sometimes impractical to enumerate all tag pairs as candidates for subsumption relation detection. To avoid brute-force comparison, we leverage the co-occurrence information to limit the number of candidates. Previous research [8], [15] shown the effectiveness of the co-occurrence information in subsumption relation detection. So, only if two tags have once co-occurred, they can be divided into the same block and can be selected as candidate pairs. Note that given a candidate tag pair (a, b) , both a subsumes b and b subsumes a will be checked in the next learning process. Given we collected 38,205 tags from Stackoverflow, there would be over one billion pairs without blocking. Only less than 3 million candidates will be retained after using the above candidate selection mechanism. It is obvious that the blocking mechanism reduces the number of candidate pairs significantly.

C. Feature Engineering

The purpose of feature engineering is to quantitatively characterize the similarities or relatedness between tags. We define six features to characterize the tag relations. The details of these features are as follows:

Lexical Feature:

1) *Lexical Feature*: Given a tag pair “asp.net” and “asp.net.mvc”, it is intuitive that they hold the subsumption relation in term of the lexical pattern. So we define a Token-based Longest Common Sub-string Asymmetric Similarity as lexical

feature by considering the length information. Then the lexical similarity between two tags a and b is computed by

$$s(a, b) = \frac{|LCS(seq(t_a), seq(t_b))|}{|seq(t_a)|} \quad (1)$$

Where $seq(t_a)$ is the word sequence of the name of the tag a , $|\cdot|$ returns the length of a word sequence, and LCS is a function to calculate the longest common sub-string sequence between two tag labels. This similarity measure captures the lexical similarity between two tags. Since we treat version number as a separated token, this metric can also capture subsumption relations between those tags and their instance versions. For example, “c++” and “c++11”.

Co-occurrence-based Features: Although the lexical feature works well, its limitation is obvious. Many tag pairs which actually hold the subsumption relations have very low lexical similarities. Thus, we also leverage the co-occurrence information to measure the semantic relatedness between tags. For example, “word2vec” and “deep-learning” do not share any lexical tokens. However, by considering the co-occurrences of them in questions, we can find that they always co-occur and may be semantically closed.

2) *Question Divergence Feature*: We define this feature to measure the co-occurrence of tags in questions based on the Normalized Google Distance [16].

$$d(Q_a, Q_b, Q) = \frac{\log(\max(|Q_a|, |Q_b|)) - \log(|Q_a \cap Q_b|)}{\log(|Q|) - \log(\min(|Q_a|, |Q_b|))} \quad (2)$$

where Q_a and Q_b are the sets of questions annotated with a and b , respectively; and Q is the set of all questions in Stackoverflow.

3) *Sentence Divergence Feature*: Stackoverflow additionally provides wiki description for each tag. The wiki description provides a more precise explanation for each tag. If two tags have co-occurred in the same sentence such as “java” and “programming language”, they may be semantically closed even if the question divergence feature of them is rather low. Inspired by this idea, we define the sentence divergence feature which aims to measure the co-occurrence of tags in sentences. These sentences are extracted from wiki descriptions of all tags. The computation of this feature $d(S_a, S_b, S)$ is similar as Equation 2, where S_a and S_b are the sets of sentences which contain a and b , respectively; and S is the set of all sentences in all wiki descriptions from Stackoverflow.

4) *Tag Divergence Feature*: Given two tags a and b , if both of them have co-occurred with tag c , they tend to hold a semantic relation. Inspired by this basic idea, we also design a tag divergence feature to measure the relatedness between tags. We compute this feature $d(T_a, T_b, T)$ as Equation 2, where T_a and T_b are the sets of tags of Q_a and Q_b , Q_a and Q_b are the sets of questions annotated with a and b , respectively; and T is the set of all tags in Stackoverflow.

Topic-based Features: The lexical feature and the co-occurrence features only capture the semantic relation in an explicit way which can not detect the implicit relations between tags. For example, most people tend not to annotate “machine learning” together with “artificial intelligence” because “artificial intelligence” is too high-level. But it is obvious that “artificial intelligence” is semantically closed to “machine

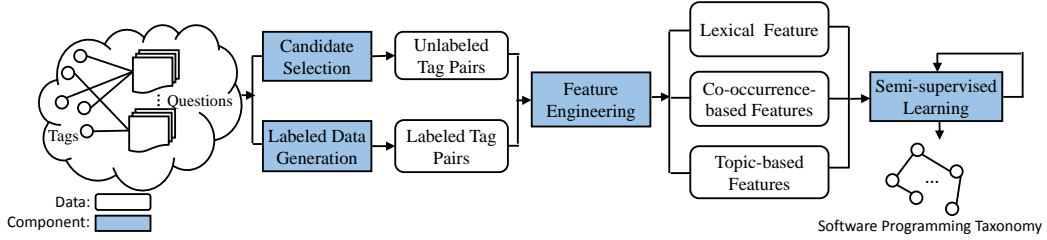


Fig. 2: The workflow to generate taxonomy for Stackoverflow

learning”. So we also try to represent each tag in a semantic level and design some topic-based features. We leverage a topic modeling method named LDA [10] to generate a topic-based representation for each tag. In our proposed approach, we model a tag using both of its wiki description and the bodies of its annotated questions, and learn its topic representation by LDA. In our experiments, the number of topics was empirically set as 150.

5) *Wiki Topic Divergence Feature*: This feature is the measure of the difference or dissimilarity between two topic distributions of wiki descriptions of tag a and tag b . We define this feature based on KL-divergence [17], a standard measure of the difference between two probability distributions. Note that it is an asymmetric metric, i.e. $d_{KL}(p_{wa}, p_{wb}) \neq d_{KL}(p_{wb}, p_{wa})$.

$$d_{KL}(p_{wa}||p_{wb}) = \sum_{i=1}^K p_{wa}(i) \log \frac{p_{wa}(i)}{p_{wb}(i)} \quad (3)$$

where $p_{wa}(i)$ and $p_{wb}(i)$ denote the probability of i -th topic in the topic distribution of a and b respectively, using wiki descriptions as document.

6) *Question Topic Divergence Feature*: In Stackoverflow, some tags only contain few contents in their wiki descriptions, especially when the tags are newly created or have fewer editors. In this circumstance, wiki topic divergence cannot accurately assess the dissimilarity between tags. However, the topic of questions can better represent their tags, since the same tag in different questions is annotated by different editors, which can avoid the subjectivity. Therefore, for a given tag, we also use questions annotated by it to represent the tag. Similarly, we compute this feature as Equation 3.

D. Labeled Data Generation

We treat subsumption relation detection as a binary-class classification problem to solve. Classification (binary or multi-class) is supervised learning, which requires labeled data for training. The classification performance depends on whether the labeled data is adequate and whether training data and test data have the similar distributions. In order to ease the burden of manual labeling and to avoid distribution bias, we propose an effective rule-based method to create labeled data. Both positive and negative examples are checked manually. These examples are not only used to boost the learning process but also treated as ground truth to evaluate our approach in Section IV.

For positive examples, we apply some lexical-syntactic patterns on descriptions of tags. These patterns are extended from the Hearst patterns [11] (e.g. NP1 is a/an NP2). Finally, we have 12,608 hypernym-hyponym candidate relations between tags and 2,870 of them are manually checked as positive examples.

For negative examples, we define a conditional probability metric to measure the probability of a as the hypernym given b . This metric relies on an implication, that if a user has annotated a document d with b , he also tend to annotate tags that subsumes b . Specifically, we use the following formula:

$$p(a|b) = \frac{N_d(a, b)}{N_d(b)} \quad (4)$$

where $N_d(a, b)$ is the number of documents that are annotated by both a and b , and $N_d(b)$ is the number of documents that are annotated by b . Given a tag b , we select the tag a as its hypernym with its probability less than 0.01 and treat the pair (a, b) as negative examples. Besides, we also manually select those relations, which do not hold subsumption relations but their probabilities $p(a|b)$ are more than 0.5 to enrich the set of negative examples. Finally, among these enriched sets, 3,000 negative examples are manually checked and selected.

E. Semi-supervised Learning

While we generate labeled data by applying some lexical-syntactic patterns and heuristic rules semi-automatically, the number of positive and negative examples is very small compared with that of the candidate tag pairs. So a natural idea is to use some kind of semi-supervised learning algorithm to predict new semantic relations between these candidate pairs.

We select the simplest and the most efficient one - self-training. In each iteration, self-training accepts the labeled data as training data and learns a classifier. Then the classifier is applied to the unlabeled data and adds tag pairs of high confidence to the labeled data to train a new classifier for the next iteration. The whole process will terminate if the difference between the predicted labels of these candidates (whether they satisfy subsumption relations or not) given by classifiers in the two consecutive iterations is smaller than a threshold or we have achieved the maximal number of iterations. Note that we use the Support Vector Machine (SVM) algorithm to train the binary classifier, which is known as one of the best single classifiers [18].

IV. EXPERIMENTS

A. Experiment Setup

1) *Data Statistics*: In order to evaluate our approach, we use the Stackexchange dump from <https://archive.org/details/stackexchange>. Note that before further process, we first singularize the names of all tags and then replace underscores and hyphens by spaces as preprocess. In total, there are 38,205 tags and 7,990,787 questions. Among these tags, 25,798 tags have descriptions. The number of questions annotated by the tag ranges from 1 to 708,533. On the average, each tag annotated 617 questions.

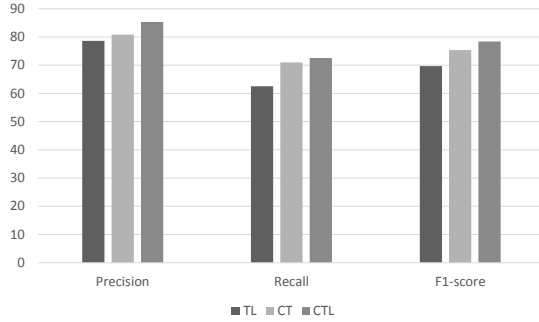


Fig. 3: Three models with different feature sets

2) *Comparison Methods*: We select several state of the art methods as comparison methods, namely Tag-Tag Co-occurrences (TTC) method and Tag-Word Co-occurrences (TWC) method.

- **Tag-Tag Co-occurrences (TTC)**. The TTC method [15] uses Equation 4 to estimate $p(a|b)$. One of its benefits is that it does not rely on the content of the annotated document, so it can be applied to tags for non-text objects.
- **Tag-Word Co-occurrences (TWC)**. This method [8] uses the content of the annotated document to estimate $p(a|b)$. We use the following formula to estimate $p(a|b)$ by tag-word co-occurrences:

$$\begin{aligned}
 p(a|b) &= \sum_{w \in W} p(a|w)p(w|b) \\
 &= \sum_{w \in W} \frac{N_d(a, w)}{N_d(w)} \frac{N_d(b, w)}{N_d(b)}
 \end{aligned} \tag{5}$$

where $N_d(a, w)$ is the number of documents that contains both tag a and word w , and $N_d(w)$ is the number of documents that contains the word w . Instead of computing tag-tag co-occurrences directly, TWC uses words in the document as a bridge to estimate $p(a|b)$.

For these two comparison methods, we first sort the discovered relations by their probabilities in descending order. Then, we take the top- n relations, discarding the others. Here we evaluate these methods with $n = 1$ and $n = 5$.

B. Result Analysis

1) *Feature Contribution Analysis*: We discuss the effect of different features for predicting subsumption relations. We use the labeled data generated in Section III-D as the ground truth. Then we train three SVM classifiers based on different combinations of features. The first classifier (denoted as *TL*) only uses topic-based features and the lexical feature. The second classifier (*CT*) combines the co-occurrence-based features and the topic-based features. The third one (*CTL*) includes all features. We apply 5-fold cross validation to train the three classifiers. Precision, recall, and F-measure are used for effectiveness study. As shown in Fig. 3, the classifier with all features performs best. That is to say, all these features are useful in predicting new subsumption relations. We can also find all classifiers use topic-based features, this is mainly because the subsumption relation is asymmetric, so only asymmetric

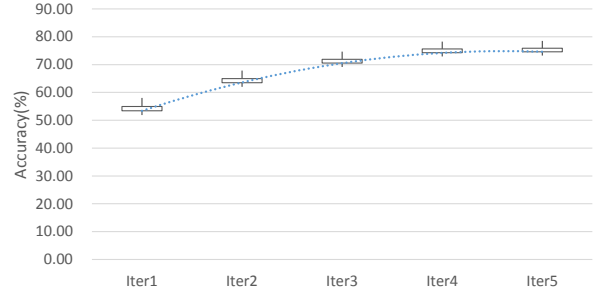


Fig. 4: Accuracy in each iteration

metrics can capture this relation semantically. In our feature set, only topic-based features can measure the subsumption relations while the co-occurrence-based features are symmetry and the lexical feature can only capture the surface patterns.

2) *Accuracy Evaluation of Iterations*: We applied the iterative semi-supervised learning approach with all features to build a taxonomy containing 68,098 subsumption relations. Since there are no ground truths available for the whole taxonomy, the qualities of these relations have to be verified manually. Due to the large number of relations, it is impossible to evaluate all of them by hand. Therefore, we design an evaluation theme including a *sampling* strategy and a *labeling* process. *Sampling* aims to extract a subset of relations (called samples) which can represent the distribution of the whole result set. Then we can perform manual labeling to evaluate the correctness of samples. The accuracy assessment on the subset can further be used to approximate the correctness of the resulting taxonomy.

We first evaluate the performance improvements in each iteration compared with the estimated accuracy produced by the previous iteration if existed. We randomly select 1,000 subsumption relations from the resulting taxonomy. We record their predicted labels after each iteration. Four students from our laboratory are invited to participant in the labeling process. We provide them three choices namely *agree*, *disagree* and *unknown* to label each sample. Then we can compute the average accuracy. Finally, the *Wilson interval* [19] at $\alpha = 5\%$ is used to generalize our findings on the samples to the whole taxonomy. Fig. 4 shows the accuracy of each iteration. According to the results, the accuracy increases consistently when we perform more iterations. In particular, after the fifth iteration, our approach achieves the best accuracy of $75.90\% \pm 2.64\%$.

C. Performance Comparison

We further compare the two baseline approaches with ours. Accuracy and scale are used as evaluation metrics. According to the result, the TTC and TWC method get a similar scale of 184,818 and 184,816 subsumption relations respectively with $n = 5$. It is obvious because they all apply a ranking mechanism by probability. For accuracy, the TTC method achieves accuracy of $53.29\% \pm 3.09\%$, while the TWC method only achieves accuracy of $37.05\% \pm 3.0\%$. Even the threshold n is set to 1, the accuracy of TTC and TWC method is only $70.42\% \pm 2.82\%$ and $43.03\% \pm 3.06\%$ respectively with the trade-off of scale which is only 38,166. Compared with these two baseline methods, our method can find 68,098 subsumption relations and the accuracy of resulting taxonomy using our proposed method is $75.90\% \pm 2.64\%$. Therefore, our approach can not

TABLE I: Comparison with other datasets

	Ours	Yago	WikiTaxonomy	Schema.org
Concept Number	38,205	898	711	10
Concept Overlap	/	29	27	2
Subsumption Number	68,098	870	630	0
Subsumption Overlap	/	0	1	0
Maximum Depth	28	3	6	1
Minimum Depth	1	2	1	1
Average Depth	6.99	2.24	1.39	1.00

only discover more subsumption relations but also achieve a better accuracy.

D. Comparison with Other Datasets

Since there is no public software programming taxonomy, we only compare our taxonomy with the subsets about software programming extracted from other well-known general-purpose datasets namely Yago Taxonomy, WikiTaxonomy and Schema.org¹ in terms of concepts and subsumption relations. Table I not only shows the concept and subsumption information of each dataset, but also lists the concept overlaps and subsumption overlaps between our taxonomy and the other datasets. Moreover, we present the maximum, minimum and average depth of each dataset to illustrate the granularity and richness. As for the concept and subsumption number, our taxonomy is much larger than any other datasets. The overlaps of both concept and subsumption with these datasets are not so high. The reason mainly comes from two aspects. First, concepts in our taxonomy are fine-grained while those in Yago Taxonomy, WikiTaxonomy and Schema.org are more high-level. Second, the overlaps between our taxonomy and any of the other three (i.e., Yago Taxonomy, WikiTaxonomy, and Schema.org) are the lower bounds and can actually be larger due to the fact that we compute the overlaps using the exact string matching and tag from Stackoverflow are not as formal as those in the three compared datasets.

Regarding to the granularity and richness of concepts, our taxonomy has the largest average depth and maximum depth. That is to say, our taxonomy has a more fine-grained concept hierarchy compared with other existing datasets. As a representative example, our taxonomy contains a hypernymy path like “machine learning”→“bayesian”→“bayesian network”→“belief propagation”. Moreover, some newly terms can also be found in our taxonomy such as “neural network”→“deep learning”→“word2vec”. So our taxonomy contains not only many newly-added and fine-grained concepts, but also a richer semantic hierarchy.

V. CONCLUSION AND FUTURE WORK

In this work, we proposed a machine learning based approach with some novel features to automatically create hypernym-hyponym relations between tags in Stackoverflow, which results in a taxonomy about software programming containing 38,205 concepts and 68,098 relations. The experiments show the high-quality of this taxonomy.

As for future work, we will try to extract more concepts about computer programming from Wikipedia, Github and other Web sites to enrich our taxonomy. Moreover, it would be interesting to explore some more potential applications based on this taxonomy such as linked data based recommendation,

semantic relatedness measuring between terms about software programming and so on.

VI. ACKNOWLEDGMENT

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242).

REFERENCES

- [1] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 15–25. ACM, 2011.
- [2] Giriprasad Sridhara, Emily Hill, Lori Pollock, and K Vijay-Shanker. Identifying word relations in software: A comparative study of semantic similarity tools. In *ICPC 2008*, pages 123–132. IEEE, 2008.
- [3] J-R Falleri, Marianne Huchard, Mathieu Lafourcade, Clementine Nebut, Violaine Prince, and Michel Dao. Automatic extraction of a wordnet-like identifier network from software. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 4–13. IEEE, 2010.
- [4] Simone Paolo Ponzetto and Michael Strube. Wikitaxonomy: A large scale knowledge resource. In *ECAI*, volume 178, pages 751–752, 2008.
- [5] Fei Wu and Daniel S Weld. Automatically refining the wikipedia infobox ontology. In *WWW*, pages 635–644. ACM, 2008.
- [6] Mianwei Zhou, Shenghua Bao, Xian Wu, and Yong Yu. *An unsupervised model for exploring hierarchical semantics from social annotations*. Springer, 2007.
- [7] Jie Tang, Ho-fung Leung, Qiong Luo, Dewei Chen, and Jibin Gong. Towards ontology learning from folksonomies. In *IJCAI*, volume 9, pages 2089–2094, 2009.
- [8] Xiance Si, Zhiyuan Liu, and Maosong Sun. Explore the structure of social tags by subsumption relations. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1011–1019, 2010.
- [9] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.
- [10] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.
- [11] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.
- [12] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q Zhu. Probbase: A probabilistic taxonomy for text understanding. In *SIGMOD 2012*, pages 481–492. ACM, 2012.
- [13] Huairan Lin, Joseph Davis, and Ying Zhou. An integrated approach to extracting ontological structures from folksonomies. In *The semantic web: research and applications*, pages 654–668. Springer, 2009.
- [14] Haofen Wang, Tianxing Wu, Guilin Qi, and Tong Ruan. On publishing chinese linked open schema. In *ISWC 2014*, pages 293–308. Springer, 2014.
- [15] Patrick Schmitz. Inducing ontology from flickr tags. In *Collaborative Web Tagging Workshop at WWW2006*, volume 50, 2006.
- [16] Rudi L Cilibrasi and Paul MB Vitanyi. The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.
- [17] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.
- [18] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.
- [19] Lawrence D Brown, T Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, pages 101–117, 2001.

¹<https://schema.org/>

An empirical study on predicting defect numbers

Mingming Chen^{1,2} and Yutao Ma^{2,3,*}

1. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China

2. School of Computer, Wuhan University, Wuhan 430072, China

3. WISET Automation Co., Ltd., Wuhan Iron and Steel Group Corporation, Wuhan 430080, China

*E-mail: ytma@whu.edu.cn

Abstract—Defect prediction is an important activity to make software testing processes more targeted and efficient. Many methods have been proposed to predict the defect-proneness of software components using supervised classification techniques in within- and cross-project scenarios. However, very few prior studies address the above issue from the perspective of predictive analytics. How to make an appropriate decision among different prediction approaches in a given scenario remains unclear. In this paper, we empirically investigate the feasibility of defect numbers prediction with typical regression models in different scenarios. The experiments on six open-source software projects in PROMISE repository show that the prediction model built with Decision Tree Regression seems to be the best estimator in both of the scenarios, and that for all the prediction models, the results yielded in the cross-project scenario can be comparable to (or sometimes better than) those in the within-project scenario when choosing suitable training data. Therefore, the findings provide a useful insight into defect numbers prediction for those new and inactive projects.

Keywords: defect prediction; predictive analytics; cross-project scenario; regression model

I. INTRODUCTION

Nowadays, defect prediction has attracted much attention from both academia and industry because of its importance in software quality assurance. It has been widely recognized that the defect-proneness of software components (such as classes and code modules) is closely related to a considerable number of software metrics (the so-called features) [1], e.g., static code metrics, code change history, process metrics and network metrics [2], all of which are easy to collect now. Therefore, many defect prediction approaches using statistical methods or machine learning techniques have been proposed to forecast defect-prone software components [3].

To the best of our knowledge, the vast majority of prior defect prediction approaches predict whether a given software component is defect-prone by means of binary classification techniques [3, 4]. However, estimating the defect-proneness of a given set of software components is not enough for software testing in practice due to plenty of criticisms of practicality [4]. Furthermore, if we are able to predict the exact number of bugs in each software component to be tested, software developers or software testers will pay special attention to those software components that contain more defects, which can make testing processes more efficient in the case of limited time and human resources. Thus, defect prediction is not just a simple binary classification problem but a specific problem with predictive analytics [5].

Due to the limitations of data analysis techniques and available training data, the focus of early studies about this topic was on building linear prediction models based on the correlations between defects and some important code features, e.g., lines of code [6] and McCabe's cyclomatic complexity. After object-oriented design metrics and process quality metrics were proposed in the 1990s, the researchers in this field developed many new prediction models by means of multiple regression analysis [7]. With the development of data mining and machine learning techniques, a few of complex prediction models for the number of defects were presented in recently published literature. For example, Wang *et al.* proposed an approach based on a defect state transition model [8]. The experimental results on open-source software projects showed that the complex models outstripped other competing models in terms of evaluation measures such as the mean absolute error, but their generality and construction cost were questioned.

Interestingly, several recent studies with respect to software defect classification [3, 9, 10] have found that simple classifiers, e.g., Naïve Bayes and Logistic Regression, were able to perform well in both within-project and cross-project scenarios, though those complex ones always achieved high precision. As we know, newly created or unpopular software projects have little historical data available to train any classifiers, which is very similar to the typical problem *cold start* in recommender systems [11]. Hence, cross-project defect prediction (CPDP) emerges as a promising solution to the above issue. Overall, it applies the prediction model learned from other selected projects to a target project [12], and the feasibility of CPDP has been widely examined by the researchers in this field [13, 14]. However, compared with within-project defect prediction (WPDP), in most cases the prediction performance of CPDP is relatively poor on account of the diversity of data distributions between source and target projects [10, 12-14].

As far as we know, very few prior studies evaluated and compared the existing approaches that predict defect numbers in different scenarios. How to make an appropriate decision among those prediction approaches in a given scenario remains unclear. Inspired by the recent studies on software defect classification, in this paper our goal is to validate the feasibility of CPDP approaches (based on regression models) to predicting defects, and to investigate which frequently-used regression model can achieve the best result in both within- and cross-project scenarios. With the pre-designed experiments that were conducted on six open-source software projects in the famous PROMISE repository, we hope our empirical findings could refine the previous work on predicting software defects.

In particular, we provide a more comprehensive and detailed suggestion about choosing appropriate predictive modeling approaches and training data to build a simple, highly cost-effective prediction model according to specific requirements.

The remainder of this paper is organized as follows: Section II introduces the related work; the research questions to be discussed are presented in Section III; Section IV and Section V show the experimental setups and results, respectively; in the end, Section VI concludes this paper and gives a research agenda for our future work.

II. RELATED WORK

Defect prediction has been an active research topic in software engineering for decades [3]. For the theme of this paper, earlier studies focused on analyzing the relationship between defects and code complexity metrics (such as lines of code) with the methods of linear regression [15]. With the rapid development of object-oriented programming and software process management techniques, some of new prediction models began to utilize more types of metrics to predict defect numbers by means of multiple regression analysis [7, 16]. According to a recent survey [1] conducted on 106 papers that were published between 1991 and 2011, the proportions of object-oriented, source code, and process metrics used are about 49%, 27%, and 24%, respectively. Radjenovic *et al.* also find that the Chidamber and Kemerer (CK) metrics are the most commonly used metrics [1]. However, the accuracy of these prediction models is not completely satisfying.

The rise of data mining and machine learning techniques fosters a few complex prediction models using random forest [17], linear discriminant analysis (LDA) [18], artificial neural networks [19], k-nearest neighbors (KNN) algorithm [20], Bayesian networks [21], support vector machines (SVM) [22], and so on. For example, Nguyen *et al.* proposed a similarity-based approach employing an improved KNN algorithm to predict defect numbers [5]. So far, they have been widely used to estimate the defect-proneness of software components, and more details of these approaches can refer to the recent surveys [3, 4]. On the other hand, considering a large number of software metrics, feature subset selection and dimensionality reduction techniques have also been applied to these new defect prediction methods [22, 23], and many empirical studies have demonstrated that they are able to achieve higher accuracy and computing efficiency by removing redundant and irrelevant software metrics [10].

Generally speaking, most of the above-mentioned studies about defect prediction are conducted in WPDP settings, because this is intuitive and easy to use. But, WPDP is not always practicable when lacking historical defect data in a given project. So, CPDP models are investigated to predict defect-prone software components according to the information or knowledge extracted from other similar software projects.

To the best of our knowledge, CPDP was first introduced to this field by Briand *et al.* [24]. They trained a prediction model according to the Xposem project, and used it to predict the defect-proneness of the Jwriter project. The experimental result showed that CPDP outperformed a random prediction model. Then, Zimmermann *et al.* [12] employed 622 cross-project

combinations among 12 open-source software projects to validate the feasibility of CPDP, but they found that only 21 out of 622 combinations worked successfully. In fact, the quality of cross-project training data, rather than the total quantity of data available from other projects, is more likely to affect the performance of CPDP models to some extent [25]. Hence, how to select the most appropriate cross-project data for a target project has recently become an interesting problem [26]. For example, Turhan *et al.* [26] applied a nearest neighbor filtering technique to filter out those irrelevant project data in the setting of CPDP, leading to a better prediction performance. More discusses on the comparison between WPDP and CPDP please refer to [4, 10, 27, 28]. Unfortunately, very few prior studies paid attention to the issue in question in CPDP settings.

III. RESEARCH QUESTIONS

Regression analysis is a commonly-used, effective method for predictive analytics, which analyzes current and historical data to make predictions on future or unknown events by estimating the relationships among different variables. In this paper, we investigate the problem related to defect numbers prediction by means of regression analysis. More specifically, we attempt to find empirical evidence to address the following two research questions:

- **RQ1:** *Which type of regression models is the most suitable approach to predicting the number of defects in different scenarios?*

As mentioned earlier, many prediction models, built with regression analysis (such as linear regression, Bayesian regression, and SVM regression), have been used to predict the number of defects. They are a group with excellence as well as shortcomings. So, we should take into consideration various factors (rather than just accuracy) when applying them to different types of actual projects with limited resources, which is required to make an optimal (or near-optimal) tradeoff among generality, performance and construction cost. That is, we want to find one or more appropriate regression models that can be used in different scenarios, because the previous studies about defect-proneness prediction have showed that the classifiers which are simple and easy to use tend to perform well in both within- and cross-project scenarios [10, 27]. In particular, is this still practicable for defect numbers prediction?

- **RQ2:** *Are the accuracies of CPDP regression models under discussion comparable to those of WPDP regression models?*

It is acknowledged that the defect prediction models trained from the same project are, in general, better than those trained from other similar projects, because different projects may have different contextual characteristics, e.g., development process, developers, and project organization. But, does it definitely happen when training data and test data have similar distributional features? So, we attempt to empirically compare the performance differences among the regression models discussed in this paper in both within- and cross-project scenarios. Note that, if the distributions of two sets of prediction results (*A* and *B*) have no statistically significant difference, in our opinion, *A* is comparable to *B*.

IV. EXPERIMENTAL SETUPS

A. Data Collection

To validate the feasibility of defect numbers prediction in the cross-project scenario, six open-source software projects with 26 releases collected from the online, publicly available PROMISE repository are used as our experimental data set. The brief introduction to all releases of the projects is shown in Table I, where *#Instances* indicates the number of instances (class files), *#Defects* denotes the total number of defects in the release, *%Defect* represents the percentage of defect-prone instances, and *Max* is the maximum value of defects.

Due to space limitations, the list of software metrics used as features in different regression models please refer to [10]. In our experiments, there are 20 independent variables (such as the CK metrics suite and LOC) and one dependent variable (the number of defects), and the goal of our experiments is to estimate the prediction results calculated using six commonly used regression models in different scenarios.

TABLE I. BRIEF INTRODUCTION TO THE EXPERIMENTAL DATA SET

Project	Release	#Instances	#Defects	%Defect	Max
Ant	Ant-1.3	125	33	16.0%	3
	Ant-1.4	178	47	22.5%	3
	Ant-1.5	293	35	10.9%	2
	Ant-1.6	351	184	26.2%	10
	Ant-1.7	745	338	22.3%	10
Camel	Camel-1.0	339	14	3.4%	2
	Camel-1.2	608	522	35.5%	28
	Camel-1.4	872	335	16.6%	17
	Camel-1.6	965	500	19.5%	28
Forrest	Forrest-0.6	6	1	16.7%	1
	Forrest-0.7	29	15	17.2%	8
	Forrest-0.8	32	6	6.3%	4
Jedit	Jedit-3.2	272	382	33.1%	45
	Jedit-4.0	306	226	24.5%	23
	Jedit-4.1	312	217	25.3%	17
	Jedit-4.2	267	106	13.1%	10
	Jedit-4.3	492	12	2.2%	2
Prop	Prop-1	18471	5493	14.8%	37
	Prop-2	23014	4096	10.6%	27
	Prop-3	10274	1640	11.5%	11
	Prop-4	8718	1362	9.6%	22
	Prop-5	8516	1930	15.3%	19
	Prop-6	660	79	10.0%	4
Synapse	Synapse-1.0	157	21	10.2%	4
	Synapse-1.1	222	99	27.0%	7
	Synapse-1.2	256	145	33.6%	9

B. Experiment Design

To answer the two research questions presented in Section III, the overall framework of our experiments and empirical analysis is shown in Figure 1.

First, for a target release (test data) of a given project, two training data selection approaches (viz. application scenarios) are considered in our experiments. (1) WPDP: all historical

releases prior to the target release within the same project are used as training data; and (2) CPDP: all available releases from the most suitable project (rather than from the same project) are used as training data. Take Ant-1.7 as an example, the four releases from the same project, namely Ant-1.3, Ant-1.4, Ant-1.5, and Ant-1.6, are selected as training data in the within-project scenario, and all of the releases from the Camel project are chosen as training data in the cross-project scenarios (see the example in Figure 1).

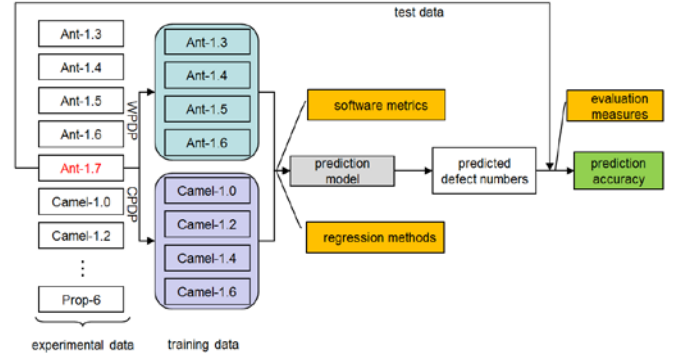


Figure 1. Overall Framework of Our Experiments: An Example of Ant-1.7

Second, we determine the number of experiments according to the experimental data and the training data selection methods. Because the first release of each project is impossible to be test data in the within-project scenario, there are 20 ($4 + 3 + 2 + 4 + 5 + 2 = 20$) groups of tests among all the releases of the six projects. To keep the comparison between WPDP and CPDP in the same condition, we select only 20 groups of corresponding tests for CPDP, though there is a total of 26 test data sets.

Third, according to the 20 software metrics (independent variables) and the number of bugs (dependent variable), we build different prediction models based on six commonly used regression methods (see the upcoming subsection) and apply them to 12 ($2 \times 6 = 12$) cases. For each case (in either WPDP or CPDP scenario) in question, we conduct the above-mentioned experiment (viz. prediction) 20 times.

Fourth, we preprocess the predicted defect numbers of all experiments before estimating the experimental results. Since the number of bugs in every software component must be a non-negative integer, we make appropriate adjustments for those original defect numbers if necessary. That is, if the predicted defect number is negative, it will be set to 0; if the result is a positive decimal, it will be set to an integer by a rounding method.

Finally, we compute the accuracy of a prediction model in terms of several evaluation measures, and compare the differences on the distributions of prediction results using statistical methods such as the Wilcoxon signed-rank test.

C. Regression Models

Regression methods assess the expectation of a dependent variable according to a set of given independent variables. More specifically, a regression model can help understand how a dependent variable changes when independent variables vary in training data sets, and tries to estimate the expectation of the dependent variable with those given independent variables in

test data sets. For the issue discussed in this paper, we assume that the vector \mathbf{X} of software metrics includes all independent variables and the numerical value of defect number y is the dependent variable. Each regression model under discussion learns a function $y = f(\mathbf{X})$ according to the relationship between \mathbf{X} and y extracted from training data, and then uses the function f to predict defect numbers of test data. Considering the motivation of this paper, we select only six typical regression models, excluding those complex ones. To avoid reinventing the wheel all the time, we build and implement these regression models based on the python machine learning library *sklearn*. Unless otherwise specified, the default parameter settings for different regression models used in our experiments are specified by *sklearn*. That is, we do not perform additional optimization for each regression model. The brief introduction to the six regression models is described as follows:

- LR (Linear Regression): it is always used to solve the least squares function of the linear relationship between one or multiple independent variables and one dependent variable.
- BRR (Bayesian Ridge Regression): it is similar to the classical Ridge Regression. The hyper parameters of such type of models are introduced by prior probability and then estimated by maximizing the marginal log likelihood with probabilistic models.
- SVR (Support Vector Regression): it is extended from the well-known Support Vector Machines, which only depends on a subset of training data, because the cost function for building a SVR model ignores any training data close to the prediction results of the model.
- NNR (Nearest Neighbors Regression): it is based on the k-nearest neighbors algorithm, and the regression value of an instance is calculated by the weighted average of its nearest neighbors. And, the weight is settled proportional to the inverse of the distance between the instance and its neighbors.
- DTR (Decision Tree Regression): it learns simple decision rules to approximate the curve of a given training data set, and then predicts the target variable.
- GBR (Gradient Boosting Regression): it is in the form of an ensemble of weak prediction models. Several base estimators are combined with a given learning algorithm in order to improve the prediction accuracy over a single estimator.

D. Evaluation Measures

To evaluate the prediction results of our experiments, in this paper we utilize the metrics precision (P) and root mean square error (RMSE), which are described as follows:

- P: Precision addresses the percentage of correctly predicted instances to the total number of test data. The higher the precision, the better accuracy a prediction model achieves.

$$P = \frac{N_{\hat{r}=r}}{N}, \quad (1)$$

where N is the number of instances in test data, and $N_{\hat{r}=r}$ indicates the number of instances whose predicted values (\hat{r}) are equal to their real values (r).

- RMSE: It measures the difference between the values predicted by a model or an estimator and the values actually observed. Compared with the mean absolute error, because of being scale-dependent, RMSE is a good measure of accuracy to compare prediction errors of different models for a given variable, e.g., the number of defects in this paper.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N |\hat{r}_i - r_i|^2}{N}}. \quad (2)$$

V. EXPERIMENTAL RESULTS

A. Answer to RQ1

First, for each prediction model built with a regression model in question, we conduct the pre-designed experiment 20 times in the scenario of WPDP. The prediction results of an example are shown in Table II, where each record in the table is denoted by a two-tuple/pair (P, RMSE). In this example, the last release of each project is used to be test data, and all of the previous releases in the same project are selected as training data. The number in bold in this table indicates the best estimator among the six prediction models. Note that BRR and GBR achieve the same value of precision, but the former is better than the latter because of a smaller RMSE value.

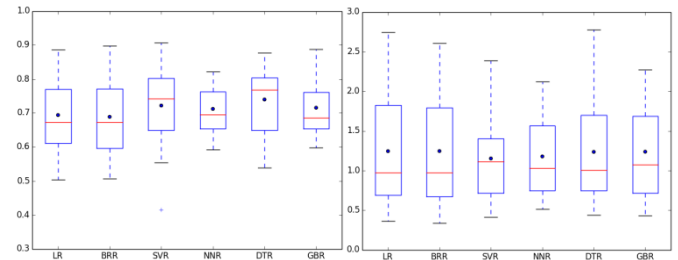


Figure 2. Standardized box plots of evaluation measures for the six prediction models in the scenario of WPDP (P: left, RMSE: right)

Moreover, the distributions of the values of P and RMSE for the 20 experiments are shown in Figure 2, where the X-axis means the prediction models under discussion and the Y-axis indicates the value of an evaluation measure. The legends from the bottom to the top of a standardized box plot are minimum, first quartile, median (red line), third quartile, and maximum. Note that the small cycle within each box is the mean value of the 20 predictions. It is obvious from Figure 2 that the median and mean of DTR are larger than those of the other five prediction models with respect to precision. On the other hand, considering the distribution of RMSE values, DTR is similar to LR and BRR, closely followed by GBR and NNR. To sum up, in the scenario of WPDP, DTR seems to be the best estimator for the experimental data in this paper.

Second, in the scenario of CPDP, we conduct the pre-designed experiment 20 times in a similar way as described for WPDP. The prediction results of an example corresponding to the above example are listed in Table III. In this example, for each target release, we select all available releases from the

most appropriate project as training data. That is, each record in this table represents the best outcome among the other projects. Surprisingly, the results about precision in Table III are, on

average, higher than those in Table II, implying that CPDP achieves better performance than WPDP in this example. This finding is different from the results of many prior studies [3, 9].

TABLE II. ESTIMATING PREDICTION RESULTSS IN THE SCENARIO OF WPDP: AN EXAMPLE

Project	LR	BRR	SVR	NNR	DTR	GBR
Ant-1.7	0.740, 0.924	0.753, 0.924	0.771 , 1.227	0.719, 0.987	0.761, 0.943	0.754, 0.921
Camel-1.6	0.616, 1.868	0.614, 1.813	0.550, 1.761	0.694, 1.775	0.739 , 1.862	0.662, 1.686
Forrest-0.8	0.750, 0.728	0.531, 0.952	0.906 , 0.810	0.812, 0.847	0.718, 1.457	0.714, 1.794
Jedit-4.3	0.597, 1.954	0.583, 1.941	0.706, 0.832	0.691, 1.651	0.798 , 1.666	0.680, 2.068
Prop-6	0.886, 0.421	0.887, 0.419	0.868, 0.447	0.809, 0.554	0.868, 0.483	0.887, 0.431
Synapse-1.2	0.632, 1.011	0.668 , 0.988	0.656, 1.208	0.636, 1.077	0.652, 1.034	0.652, 1.019

TABLE III. ESTIMATING PREDICTION RESULTS IN THE SCENARIO OF CPDP: AN EXAMPLE

Project	LR	BRR	SVR	NNR	DTR	GBR
Ant-1.7	0.762, 1.016	0.764, 1.015	0.756, 1.132	0.745, 1.128	0.756, 1.132	0.775 , 1.113
Camel-1.6	0.771, 1.752	0.773, 1.770	0.842 , 1.599	0.754, 1.807	0.778, 1.795	0.764, 1.145
Forrest-0.8	0.931, 0.780	0.938 , 0.791	0.894, 0.792	0.875, 0.829	0.927, 0.740	0.786, 1.804
Jedit-4.3	0.821, 0.666	0.823, 0.664	0.749, 0.687	0.829, 0.813	0.880, 0.486	0.896 , 0.789
Prop-6	0.779, 0.529	0.826, 0.488	0.891 , 0.421	0.818, 0.486	0.885, 0.478	0.849, 0.478
Synapse-1.2	0.668, 1.084	0.648, 1.046	0.711 , 1.240	0.648, 1.137	0.684, 1.093	0.664, 1.108

Similarly, Figure 3 shows that in the scenario of CPDP DTR is also the best estimator when considering precision. With regard to RMSE, DTR is similar to LR, which is next to BRR. Note that any data not included between the whiskers is plotted as a cross in Figure 3.

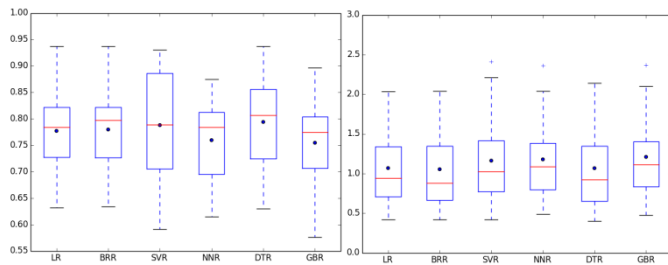


Figure 3. Standardized box plots of evaluation measures for the six prediction models in the scenario of CPDP (P: left, RMSE: right)

The prediction model based on Decision Tree Regression is the best estimator for defect numbers in different scenarios.

B. Answer to RQ2

In both within- and cross-project scenarios, the results of a prediction model are actually two groups of samples, and the goal of RQ2 is to assess whether their population mean ranks differ. The Wilcoxon signed-rank test, also known as a non-parametric statistical hypothesis test, is suitable for this case, because the population cannot be assumed to be normally distributed (see Figures 2 and 3, the median and mean within each box are not equal). The software program for such tests is realized based on *scipy*, which is a Python-based software ecosystem for mathematics, science, and engineering.

Assuming that two groups of samples are drawn from the same distribution (*null hypothesis*), the Wilcoxon signed-rank test is executed with an alternative/opposite hypothesis. A *p*-value generated in the test is used to reject the *null hypothesis* in favor of the opposite hypothesis. But, if the *p*-value is more than 0.01, one cannot reject the *null hypothesis*. The test results are shown in Table IV, which highlights that there are no

significant differences among WPDP and CPDP models, indicated by the majority of $p > 0.01$ (10/12) for these models evaluated by the two measures, though two exceptions (whose numbers are in bold font) exist with respect to precision. More interestingly, for the two exceptions, the Cliff's effect sizes for NNR and DTR are negative, which suggests that the results of CPDP models are better. From the perspective of statistical analysis, the finding indicates that in this paper CPDP models are comparable to (or sometimes better than) WPDP models with regard to accuracy, largely due to that in some cases we select a large and mature project (Prop) as training data in the scenario of CPDP. Hence, the selection of historical data from those suitable mature projects to train a prediction model for defect numbers is a significant factor for the success of CPDP.

TABLE IV. A COMPARISON OF THE DISTRIBUTIONS OF PREDICTION RESULTS IN BOTH SCENARIOS USING THE WILCOXON SIGNED-RANK TEST

Measure	LR	BRR	SVR	NNR	DTR	GBR
Precision	0.011	0.044	0.030	0.002	0.001	0.033
RMSE	0.314	0.108	0.941	0.970	0.084	0.296

CPDP models are comparable to (or sometimes better than) WPDP models with respect to prediction performance.

C. Threats to Validity

Although we obtain some interesting findings to answer the research questions, there are still potential threats to the validity of our work, one of which concerns the generalization of the results obtained in this paper. The main reasons include the following four aspects: (1) we randomly select six projects (a very small subset of all of the projects) from the PROMISE repository when conducting the experiments; (2) we only use the 20 static code metrics when building prediction models, because the six projects do not include the information of process metrics and social network measures [29]; (3) we select training data in a simple way, though there are many time-consuming but effective selection methods [30]; and (4) we only utilize six typical regression methods without additional optimization for a given data set.

VI. CONCLUSION AND FUTURE WORK

Defect numbers prediction is an interesting problem in the field of defect prediction. Compared with the prior studies on defect classifiers (viz. supervised classification models), in this paper we empirically investigate the feasibility of defect numbers prediction using regression methods in both within-project and cross-project defect scenarios. The experiments on six open-source software projects show that those simple, typical regression methods are also able to perform well in different scenarios, e.g., the prediction model built with Decision Tree Regression is proven to be the best estimator in our experiments, and that the six prediction models under discussion can achieve similar (or sometimes even better) results in both within- and cross-project scenarios. The findings would provide useful empirical evidence for software developers or software testers to choose appropriate training data and regression methods in the case of urgent deadlines and limited resources.

Our future work will further investigate the issues related to defect numbers prediction so as to improve prediction precision, including (1) building the best prediction model according to the actual distribution of defects in a given software project, and (2) selecting the most suitable training data for defect numbers prediction using transfer learning techniques [31] in the scenario of CPDP.

ACKNOWLEDGMENT

This work is supported by the National Basic Research Program of China (973 Program) (No. 2014CB340401), the National Natural Science Foundation of China (Nos. 61272111 and 61273216), the Youth Chenguang Project of Science and Technology of Wuhan City in China (No. 2014070404010232), and the open foundation of Hubei Provincial Key Laboratory of Intelligent Information Processing and Real-time Industrial System in China (No. znss2013B017).

REFERENCES

- [1] D. Radjenović, M. Heričko, R. Torkar, A. Živković, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, 2013, 55(8): 1397–1418.
- [2] A. Meneely, L. Williams, W. Snipes, J. Osborne, "Predicting failures with developer networks and social network analysis," in: *Proc. of the 16th ACM SIGSOFT FSE*, Atlanta, Georgia, USA, 2008, pp. 13–23.
- [3] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, "A systematic review of fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, 2012, 38 (6): 1276–1304.
- [4] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, 2015, 27: 504–518.
- [5] T.T. Nguyen, T.Q. An, V.T. Hai, T.M. Phuong, "Similarity-based and rank-based defect prediction," in: *Proc. of the 2014 Int'l Conf. on Adv. Technol. for Commun.*, Hanoi, Vietnam, 2014, pp. 321–325.
- [6] J.E. Gaffney Jr., "Estimating the Number of Faults in Code," *IEEE Transactions on Software Engineering*, 1984, 10(4): 459–465.
- [7] N.E. Fenton, M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, 1999, 25(5): 675–689.
- [8] J. Wang, H. Zhang, "Predicting defect numbers based on defect state transition models," in: *Proc. of the 6th ACM-IEEE Int'l Symp. on Empirical Softw. Eng. and Measurement*, Sweden, 2012, pp. 191–200.

- [9] D.M. Ambros, M. Lanza, R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Softw. Eng.*, 2012, 17(4-5): 531–577.
- [10] P. He, B. Li, X. Liu, J. Chen, Y.T. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, 2015, 59: 170–190.
- [11] A.I. Schein, A. Popescul, L.H. Ungar, D.M. Pennock, "Methods and metrics for cold-start recommendations," in: *Proc. of the ACM SIGIR SIGIR'02*, Tampere, Finland, 2002, pp. 253–260.
- [12] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in: *Proc. of the ESEC/FSE'09*, Netherlands, 2009, pp. 91–100.
- [13] Z. He, F. Shu, Y. Yang, M.S. Li, Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Autom. Softw. Eng.*, 2012, 19(2): 167–199.
- [14] F. Rahman, D. Posnett, P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in: *Proc. of the 20th ACM SIGSOFT FSE*, Cary, NC, USA, 2012, p. 61.
- [15] H.Y. Zhang, "An Investigation of the Relationships between Lines of Code and Defects," in: *Proc. of the 25th IEEE Int'l Conf. on Softw. Maintenance*, Edmonton, Alberta, Canada, 2009, pp. 274–283.
- [16] F. Lanubile, A. Lonigro, G. Visaggio, "Comparing models for identifying fault-prone software components," in: *Proc. of the 7th Int'l Conf. on Softw. Eng. and Knowl. Eng.*, USA, 1995, pp. 312–319.
- [17] A. Kaur, R. Malhotra, "Application of random forest in predicting fault-prone classes," in: *Proc. of the ICACTE'08*, Thailand, 2008, pp. 37–43.
- [18] J.C. Munson, T.M. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Trans. Softw. Eng.*, 1992, 18(5): 423–433.
- [19] A. Kaur, P. Sandhu, A. Bra, "Early software fault prediction using real time defect data," in: *Proc. of the 2nd Int'l Conf. on Machine Vision*, Dubai, UAE, 2009, pp. 242–245.
- [20] G.D. Boetticher, "Nearest neighbor sampling for better defect prediction," in: *Proc. of the IEEE PROMISE'05*, Missouri, USA, 2005, pp. 1–6.
- [21] G. Pai, J. Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," *IEEE Transactions on Software Engineering*, 2007, 33(10): 675–686.
- [22] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, 2008, 34(4): 485–496.
- [23] K. Gao, T.M. Khoshgoftaar, H. Wang, N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Softw., Pract. Exper.*, 2011, 41(5): 579–606.
- [24] L.C. Briand, W.L. Melo, J. Wüst, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE Trans. Softw. Eng.*, 2002, 28(7): 706–720.
- [25] P. He, B. Li, D. Zhang, Y.T. Ma, "Simplification of Training Data for Cross-Project Defect Prediction," *CoRR*, abs/1405.0773, 2014.
- [26] B. Turhan, T. Menzies, A. Bener, J.D. Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, 2009, 14(5): 540–578.
- [27] F. Zhang, A. Mockus, I. Keivanloo, Y. Zou, "Towards building a universal defect prediction model," in: *Proc. of the IEEE MSR'14*, Hyderabad, India, 2014, pp. 182–191.
- [28] B. Turhan, A.T. Misirli, A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Information and Software Technology*, 2013, 55(6): 1101–1118.
- [29] T. Zimmermann, N. Nagappan, "Predicting defects using network analysis on dependency graphs," in: *Proc. of the 30th Int'l Conf. on Softw. Eng.*, Leipzig, Germany, 2008, pp. 531–540.
- [30] F. Peters, T. Menzies, A. Marcus, "Better cross company defect prediction," in: *Proc. of the 10th Workshop on Mining Software Repositories*, San Francisco, CA, USA, 2013: 409–418.
- [31] Y. Ma, G. Luo, X. Zeng, A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, 2012, 54(3): 248–256.

Causes of Architecture Changes: An Empirical Study through the Communication in OSS Mailing Lists

Wei Ding^{1,4}, Peng Liang^{1*}, Antony Tang², Hans van Vliet³

¹ State Key Lab of Software Engineering, School of Computer, Wuhan University, China

² Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia

³ Department of Computer Science, VU University Amsterdam, The Netherlands

⁴ Key Laboratory of Earthquake Geodesy, Institute of Seismology, China Earthquake Administration, China
tingwhere@whu.edu.cn, liangp@whu.edu.cn, atang@swin.edu.au, hans@cs.vu.nl

Abstract—Understanding the causes of architecture changes allows us to devise means to prevent architecture knowledge vaporization and architecture degeneration. But the causes are not always known, especially in open source software (OSS) development. This makes it very hard to understand the underlying reasons for the architecture changes and design appropriate modifications. Architecture information is communicated in development mailing lists of OSS projects. To explore the possibility of identifying and understanding the causes of architecture changes, we conducted an empirical study to analyze architecture information (i.e., architectural threads) communicated in the development mailing lists of two popular OSS projects: Hibernate and ArgoUML, verified architecture changes with source code, and identified the causes of architecture changes from the communicated architecture information. The main findings of this study are: (1) architecture information communicated in OSS mailing lists does lead to architecture changes in code; (2) the major cause for architecture changes in both Hibernate and ArgoUML is preventative changes. (3) more than 45% of architecture changes in both projects happened before the first stable version was released, which indicates that the architectures of the investigated OSS projects are relatively stable after the first stable release.

Keywords—architecture change; cause of change; open source software; mailing list; communication

I. INTRODUCTION

Software architecture (SA) represents “*the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*” [1]. Systems continuously evolve and change to be adapted to new uses, just as buildings change over time [2], which consequently leads to architecture changes. Understanding the causes of architecture changes is important to help practitioners to understand the knowledge of the design decisions that lead to the architecture changes [3], and also allows researchers to devise means to prevent architecture knowledge vaporization and architecture degeneration [4]. The causes of architecture changes are regarded as an essential element of architectural design decision, which is a first-class entity to represent architecture [5], and are used to develop related methods to deal with specific architecture changes, for example, architects analyze due to what cause the property of an

architecture is inhibited in order to transform the architecture to satisfy non-functional requirements [6]; architectural styles as analysis tools are used to analyze the causes of architecture changes, and in turn to predict the effect of the architecture changes [7]. Architectural knowledge vaporization (e.g., design decisions and causes of architecture changes) will lead to increased maintenance costs [5]. To prevent this problem, developers (especially architects) need a way to record and communicate the causes of changes in architecture. With an explicit description of architecture as well as their changes [8], software maintainers can better understand the ramification of architecture changes and thereby more accurately analyze the impact and estimate costs of modifications [9]. But the reality is that the rationale of architectural design decisions (e.g., their causes) is often not available in SA documentation [10], especially in OSS development when SA is rarely documented (only 5.4% of 2000 investigated OSS projects have some SA documentation) [11]. We conjecture that causes of architecture changes are communicated between developers through various media, especially in a distributed development context when face-to-face communication is difficult. Mailing list is an important social media for knowledge sharing between knowledge providers and knowledge seekers in OSS projects [12]. Our recent study has shown that communication on architecture does exist in the mailing lists of two popular OSS projects (Hibernate and ArgoUML) [13], and OSS development mailing lists may act as a potential source to extract and identify the cause information of architecture changes in a project.

One of the characteristics of many successful OSS projects is the existence of a SA [14]. Architecture change is also a widespread phenomenon in OSS development, for example, an investigation of the changes in Linux kernel’s evolution indicates that most remarkable growth for a “stable” version has been in the addition of new features and support for new architectures rather than fixing defects [15]. To understand the causes of architecture changes [16][17], we conducted an empirical study to extract, identify, and analyze the architecture change information communicated in the OSS mailing lists of two popular OSS projects: Hibernate and ArgoUML based on the data (i.e., architectural threads, which are a set of communication posts on the same topic that contain architecture information in mailing lists) we collected in [13]. The identified architecture changes in

* Corresponding author

This work is sponsored by the NSFC under Grant No. 61170025, 61472286.

(DOI reference number: 10.18293/SEKE2015-193)

mailing lists were further located and verified (confirmed) in the source code of the two projects, and the causes of the architecture changes were classified through the communicated content in architectural threads. The goal of this work is to provide a practical understanding of the causes of architecture changes through communication in OSS mailing lists: Does architecture communication in mailing lists lead to architecture changes in source code? What types of causes that lead to the architecture changes? When do OSS developers communicate the causes of architecture changes?

To answer these questions, we first extracted architecture change information from the architectural threads of OSS mailing lists and further classified the causes of architecture changes with a top-down approach (i.e., using an existing categorization of causes of architecture changes provided in [16]), then checked and verified these changes against source code. We conducted this study based on the architectural threads collected in two popular OSS projects: Hibernate and ArgoUML [13], in which we identified 131 architectural threads from 20,413 posts in the mailing list of Hibernate from Jan 2002 to Aug 2014; and 200 architectural threads from 26,439 posts in the mailing list of ArgoUML from Jan 2001 to Aug 2014. These architectural threads in Hibernate and ArgoUML are used to extract, identify, and analyze the causes of architecture changes. The results show that the major cause for architecture changes in both Hibernate and ArgoUML is **preventative changes**, which ease future maintenance by restructuring or reengineering the system.

The rest of this paper is organized as follows. A brief review of related work is discussed in Section II. The methodology, including research questions and study process, is described in Section III. The results of this study are presented and discussed in Section IV. Threats to validity are discussed in Section V. We conclude and outline the future directions of this work in Section VI.

II. RELATED WORK

A. Cause of Architecture Change

The causes of architecture changes have been explored in software development in various perspectives. The work in [16] uses a systematic literature review to characterize architecture changes from existing literatures. As part of the Software Architecture Change Characterization Scheme (SACCS), a general classification of causes of architecture changes presented in [16] and another work [17] by the same authors can be used as the basic categorization of the causes of architecture changes in the two OSS projects in this study, which is elaborated in Section III. The work in [18] analyzes group interviews in various workshops for different levels of participants, e.g., developers, testers, and architects in five companies. The results validated a taxonomy of the causes for architecture technical debt, a kind of architecture inconsistency, which can be incurred and repaid by architecture changes. The work in [19] uses various versions of an ATM Simulator to observe and analyze what happens when a system evolves and new requirements are added. The results of this work show that changes in requirements may

lead to architecture changes and drift, and consequently developers (architects) that do not fully understand the design may take sub-optimal decisions, resulting in design erosion. The authors also identified the causes of design erosion, which can also be the causes for architecture change.

B. Communication through Mailing Lists in OSS Projects

Mailing lists in OSS development, as a rich source of communication of development, have been investigated in many studies. The work in [12] discusses the altruistic sharing of knowledge between knowledge providers and knowledge seekers in the developer and user mailing lists of Debian project. The authors developed the Knowledge Sharing Model (KSM) to show how knowledge can be shared (communicated) in OSS mailing lists, and used email exchanges between mailing list participants as quantifiable measures of knowledge sharing activities in OSS development. Some keywords in the subject of posts of mailing lists are used to identify posting and replying posts, e.g., “Re:”. The study in [20] examined the first posts of newcomers in the mailing lists of four popular OSS projects: MediaWiki, GIMP, PostgreSQL, and Subversion. The authors found that knowledge communication (nearly 80% of newbie posts received replies) was positively correlated with their future participation. Mockus and his colleagues used email archives of source code change history and problem reports to quantify aspects of developer participation, core team size, code ownership, productivity, defect density, and problem resolution intervals, for two large OSS projects, Apache and Mozilla [21]. These works pay attention to all the posts and threads in a mailing list during a certain period, while our study specifically extracts, identifies, and analyzes architecture changes and their causes through the communication in mailing lists.

III. METHODOLOGY

To explore the causes of architecture changes through the communication in OSS mailing lists, we select and analyze the mailing lists of two popular OSS projects: Hibernate and ArgoUML, based on the data (i.e., architectural threads) collected in our recent work [13]. In this section, we describe the design of this study with following components: the objective and research questions are presented in Section III.A, the selection criteria of the OSS projects are described in Section III.B, and the study process is elaborated in Section III.C.

A. Objective and Research Questions

The goal of this study, formulated using the Goal-Question-Metric (GQM) approach [22] is: *to analyze architecture changes through the communication in mailing lists for the purpose of characterizing the causes of architecture changes from the point of view of OSS developers in the context of OSS development*. We formulate the following research questions (RQs) based on the abovementioned goal.

RQ1: What are the causes that lead to architecture changes in OSS development?

Rationale: Mailing lists have been used as a major vehicle for the communication in OSS development [23]. Architecture information is communicated in the mailing lists of OSS projects [13]. Some of them may discuss specific architecture information, e.g., the causes of architecture changes. With the existing categorization of architecture changes provided in [16], we want to understand in a practical perspective the causes of architecture changes in OSS development through the communicated content extracted from architectural threads. Knowledge and understanding about the causes of architecture changes (evolution) as well as their risks can facilitate the development of strategies to mitigate these risks in software evolution [24].

RQ2: What are the trends of causes that lead to architecture changes in a time perspective?

Rationale: We intend to identify when various types of architecture changes happened and their causes were communicated in a time perspective. The answer of this question would allow us to further identify the best timing for performing treatments to various types of causes of architecture changes, and help practitioners to understand distribution of various causes of the changes in the development lifecycle. To investigate the relationship between causes of architecture changes and time point of releases, the studied period of both OSS projects is divided into two stages according to their first stable releases, i.e., ArgoUML v0.10¹ and Hibernate v1.0 final².

B. Selection Criteria of OSS Projects

Three criteria are used in this study to select OSS projects that have mailing lists: (1) The duration of the project is more than 10 years. (2) There are more than 1000 posts in the development mailing list of the project, which provides rich data to mine architecture information. (3) There are more than 50 developers who ever used the mailing list, which is meaningful to analyze the behavior of the developers on communicating architecture information using the mailing list.

Based on these selection criteria, we chose Hibernate and ArgoUML as the OSS projects which mailing lists were analyzed. Hibernate provides an Object/Relational mapping (ORM) framework which implements the Java Persistence API, and is popularly used in Java applications. Hibernate has 20,413 posts in its development mailing list between Jan 2002 and Aug 2014, when the release version was evolved from v0.9.1 to v4.3.6. Note that, the mailing list of Hibernate was migrated from Sourceforge to JBoss in Aug 2006. The mailing list of Hibernate in Sourceforge was initially maintained by a core developer, but he did not continue this administration effort after 11-Aug-2006³. Hence, the investigated time period of the mailing list of Hibernate in this study covers the time period of two mailing lists hosted at Sourceforge and JBoss respectively. ArgoUML is a popular open source UML modeling tool developed in Java.

¹ <http://argouml.tigris.org/servlets/NewsItemView?newsItemId=128>

² <http://sourceforge.net/p/hibernate/mailman/message/5497028/>

³ <http://sourceforge.net/p/hibernate/mailman/message/13328372/>

ArgoUML accumulates 26,439 posts in its development mailing list between Jan 2001 and Aug 2014, when the release version was updated from v0.8 to v0.34. The information of the Hibernate and ArgoUML development mailing lists are elaborated in TABLE I.

TABLE I. HIBERNATE AND ARGOUML DEVELOPMENT MAILING LISTS

OSS Project	Mailing list URL	Time period	Num. of Posts
Hibernate	http://sourceforge.net/p/hibernate/mailman/hibernate-devel/	Jan 2002 - July 2006	8,913
	http://lists.jboss.org/pipermail/hibernate-dev/	Aug 2006 - Aug 2014	11,500
ArgoUML	http://argouml.tigris.org/ds/viewForumSummary.do?dsForumId=450	Jan 2001 - Aug 2014	26,439

The IEEE 1471-2000 standard suggests ten main architecture elements for architectural description [25], which were employed as the categorization of architecture elements in our prior work to identify various architecture elements documented in an architecture document [11]. In this study, we also use this categorization to identify various architecture information communicated in mailing lists.

C. Study Process

This study is conducted in three phases. The **first phase** is data collection. We first identify the architectural threads in the mailing lists from the two OSS projects selected in Section III.B. We then extract and classify the causes of architecture changes in the following steps:

Step1: Select 30 architectural threads as the data for a pilot study;

Step2: Identify architectural threads that lead to architecture changes by checking and verifying source code;

Step3: Extract architecture changes and further classify the causes of architecture changes with a top-down approach (i.e., following an existing categorization provided in [16]);

Step4: Review the identified and classified types of causes of architecture changes by two researchers to partially mitigate the threat of personal bias;

Step5: Repeat **Step1** to **Step4** on the rest architectural threads in the mailing lists of Hibernate and ArgoUML.

The **second phase** is verification of architecture changes in source code. The *Classes*, *Packages*, or architecture design discussed in mailing lists are checked and located in source code. A semi-automatic static code analysis tool Understand⁴ is used to identify the changes of source code. Understand can identify the existence of a specific *Class* or *Package* in an Understand project compiled with OSS source code by searching with the name of the *Class* or *Package*. The difference between continuous releases in source code can be used to locate and verify the changes of architecture. For example, if a new *Class* discussed in an architectural thread appears in a certain release, e.g., v1.0, but does not exist in a previous version, e.g., v0.8, we can confirm that the

⁴ <https://scitools.com/>

communication in this architectural thread caused an architecture change (i.e., adding the new *Class*).

The **third phase** is data analysis. Qualitative and quantitative data of architecture changes are extracted from architectural threads, and used to answer the research questions. We manually checked architecture changes discussed in each architectural thread in the mailing lists and recorded the causes of architecture changes identified in the threads in an Excel spreadsheet for further analysis.

IV. RESULTS AND DISCUSSION

A. Cause of Architecture Change

Using a top-down approach by analyzing the extracted architecture changes, we identified the categories of causes of architecture changes. Architecture changes can be classified in different perspectives. Cause of architecture change is one of them. A recent literature review specifies four categories of causes for architecture changes [16]: (1) **Perfective changes** result from new or changed requirements. These changes improve the system to better meet user needs. (2) **Corrective changes** occur in response to defects in software products. (3) **Adaptive changes** occur when moving to a new environment, platform, or accommodating new standards. (4) **Preventative changes** ease maintenance by restructuring/reengineering the system.

We intend to identify whether architecture communication in mailings lists lead to architecture changes by checking source code in continuous releases. For example, when a new *Class* is suggested by a developer in a mailing list, we will check the name of this *Class* in the following releases and verify whether this *Class* is added or not. If the answer is “Yes”, we can further extract the cause of this architecture change from the discussion in the architectural thread of the mailing list. The extracted causes of architecture changes can be directly mapped to the abovementioned four categories of causes for architecture changes with the top-down approach (i.e., following an existing categorization provided in [16]). The percentages of the four types of causes of architecture changes in Hibernate and ArgoUML are showed in TABLE II.

TABLE II. PROPORTIONS OF FOUR TYPES OF CAUSES OF ARCHITECTURE CHANGES IN HIBERNATE AND ARGOUML

OSS Project	Perfective changes	Corrective changes	Adaptive changes	Preventative changes
Hibernate	25.7%	5.7%	20.0%	48.6%
ArgoUML	43.3%	10.8%	2.7%	45.9%

The proportions of the four types of causes of architecture changes before and after the first stable releases of Hibernate and ArgoUML are showed in TABLE III. The abbreviations BFR and AFR represent two stages as described in the rationale of RQ2, i.e., before and after the first stable version was released. Note that, the sum of the percentages of ArgoUML shown in TABLE II and TABLE III exceeds 100%, because one architecture change may be caused by several types of reasons (e.g., adaptive change and perfective change).

TABLE III. PROPORTIONS OF FOUR TYPES OF CAUSES OF ARCHITECTURE CHANGES BEFORE AND AFTER THE FIRST STABLE RELEASE IN HIBERNATE AND ARGOUML

OSS Project	Perfective changes	Corrective changes	Adaptive changes	Preventative changes
BFR Hibernate v1.0 (45.7%)	17.1%	5.7%	0.0%	22.9%
AFR Hibernate v1.0 (54.3%)	8.6%	0.0%	20.0%	25.7%
BFR ArgoUML v0.10 (89.1%)	35.1%	8.1%	2.7%	43.2%
AFR ArgoUML v0.10 (13.5%)	8.1%	2.7%	0.0%	2.7%

Answer to RQ1: There are four types of causes of architecture changes in OSS development: perfective changes, corrective changes, adaptive changes, and preventative changes, which cover all the types of causes of architecture changes in [16]. As shown in TABLE II, the major cause for architecture changes in both Hibernate and ArgoUML is preventative changes.

Answer to RQ2: Perfective changes and preventative changes are the main causes of architecture changes before the first stable releases in both Hibernate and ArgoUML. As shown in TABLE III, after the first stable version was released, the causes of architecture changes of Hibernate are mixed, e.g., adapted to JDK5 (adaptive change) and redesigning Hibernate to be more event-oriented (preventative change); the causes of architecture changes of ArgoUML are mostly perfective changes, e.g., adding a data interface for a new component.

B. Discussion of Study Results

Categorization of causes of architecture changes: All the causes of architecture changes in Hibernate and ArgoUML can be mapped to the four categories of causes of architecture changes provided in [16] and no new category was identified, which empirically validates that this existing categorization does work with OSS projects.

Stable and maintainable architecture: As shown in TABLE III, 45.7% architecture changes in Hibernate and 89.1% architecture changes in ArgoUML happened before the first stable version. It implies that the major part of the architectures was formed and became stable in the initial stage of the two projects. As illustrated in TABLE II, preventative changes are the major cause for architecture changes in both projects. It is not a surprising result. Preventative changes are made to easy future maintenance and evolution. OSS developers tend to make preventative changes (anticipation) in order to achieve a maintainable and evolvable architecture (e.g., refactoring architecture design to be prepared for new or changed requirements). Corrective and adaptive changes are in a small proportion in all architecture changes. The reasons are diverse, potential defects and environmental changes can be prevented and mitigated through preventative changes, or corrective changes are communicated in other sources (e.g., JIRA).

Role of core developers: Core developers refer to those that are actively involved in high levels of communication and knowledge sharing in development [26] (i.e., architecture information communication in this work), e.g.,

GK⁵ in Hibernate and AC in ArgoUML. In this study, we find that 68.5% architecture changes are made by the top two core developers in Hibernate, and 75.7% architecture changes are conducted by the top three core developers in ArgoUML, according to the core developers identified in [13]. These results show that core developers dominate the changes of architecture. These core developers also act as the role of architect in the two OSS projects.

C. Implications for Researchers and Practitioners

For researchers: The results of this study empirically show that architecture communication in OSS mailing lists is correlated with architecture changes in source code. One of the merits of the architecture information communicated in mailing lists is that it contains rich design rationale information about architecture (e.g., cause information about architecture changes), which is particularly useful to enrich architectural design decisions [27] and architecture documentation [28]. Another promising benefit of architecture changes classification in a cause perspective is that it allows researchers to develop a common approach to deal with the changes with similar causes, instead of addressing each change individually (e.g., the purpose of requirements changes classification [29]). To support the approach, a tool for addressing various types of architecture changes can be developed, e.g., certain architecture changes are better addressed by resolving their conflicts with related design decisions [19].

For practitioners: Participants of OSS projects may use the study results to guide them to trace architecture changes from mailing lists. As we have discussed in Section IV.B, architecture changes frequently happened before the first stable version was released. For example, if a new developer of an OSS project wants to get the basic knowledge about the architecture design in order to have a preliminary understanding of the system, s/he can check the architectural threads that suggest, negotiate design, and interpret design implementation through the mailing list during the early stage of development before the first stable release.

V. THREATS TO VALIDITY

The threats to the validity of this study are discussed according to the guidelines in [30].

Construct validity in this study focuses on whether we extracted architecture information from the mailing lists, identified architecture changes, and interpreted the results of this study correctly. To mitigate the bias on architecture information definition, we chose the architectural description model in IEEE Standard 1471-2000 [25] as a benchmark model to identify the threads that contain architecture information in mailing lists. To identify architecture changes, we compare the changed *Classes/Packages* between continuous releases in source code using Understand tool, which mitigates the bias on confirming architecture changes. There is a risk that the results of this study might be affected

⁵ Only the abbreviations of the developers' names are provided due to the privacy concern.

by the interpretation of the criteria for extracting and identifying architecture information and architecture changes by different researchers. A pilot data extraction was conducted by two researchers to mitigate the bias on understanding and identifying architecture changes. We admit that some causes of architecture changes at the system level are too abstract to be verified in source code by the identification method used in this study. This threat can be mitigated with an understanding of the code structure through the communication with core developers (architects).

Internal validity focuses on the avoidance of confounding factors that may influence the interpretation of the results of a study. There is a risk that the scope of architecture changes might be affected by the identification method used in this work. To mitigate this potential issue, we used the changed *Classes/Packages* to identify the architecture changes in source code by Understand tool. Some architecture changes at the system level discussed in architectural threads were excluded from analysis, because they are too abstract and we could not verify them in source code. We employed a descriptive statistics method to present the results of this study, and the threats to internal validity are minimized. We did not intend to establish any causal relationship between architecture changes and other aspects (e.g., change time) of OSS development in this study.

External validity refers to the degree to which our findings from this study can be generalized. In order to improve the generalizability of the study results and findings, we chose two popular and representative OSS projects that have mailing lists based on the selection criteria in Section III.B. Studying the mailing lists of more OSS projects based on the selection criteria can also alleviate this threat.

Reliability focuses on whether the study yields the same results if other researchers replicate it, which in this work is related to the collection and analysis of architecture changes as well as their causes. By making explicit the process and criteria of data collection and data analysis of this study in Section III, and using Understand (a third-party tool) for verifying architecture changes, this threat is mitigated.

VI. CONCLUSION AND FUTURE WORK

In this empirical study, we analyzed the architecture information communicated in architectural threads of the mailing lists of two popular OSS projects: Hibernate and ArgoUML, and located and verified architecture changes by comparing the differences between the source code of continuous OSS releases. Four types of architecture changes in a cause perspective are classified. The main findings of this work are: (1) architectural information communicated in OSS mailing lists does lead to architecture changes in code; (2) the major cause for architecture changes in both Hibernate and ArgoUML is preventative changes. (3) more than 45% of architecture changes in both projects happened before the first stable version was released, which indicates that the architectures of the investigated OSS projects are relatively stable after the first stable release.

The results of this study provide several promising research directions: (1) The results and findings of this work

can be further validated through a survey with the core developers of the two OSS projects; (2) As we mentioned in Section IV.C, a tool for a certain type of causes of architecture changes can be developed to deal with similar architecture changes based on the results of this work; (3) Other sources in OSS development, e.g., forums, commit data [31], and blogs [32], may also contain information about architecture changes and their causes. We may explore the possibility to identify architecture changes and their causes from these sources.

REFERENCES

- [1] ISO/IEC/IEEE, ISO/IEC/IEEE Std 42010-2011 International Standard, Systems and software engineering - architecture description, 2011.
- [2] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture", *Software Engineering Notes*, ACM, vol. 17, no. 4, pp. 40-52, 1992.
- [3] J. Bosch, "Software architecture: The next step", in: *Proceedings of the 1st European Workshop of Software Architecture (EWSA)*, St Andrews, UK, Springer, pp. 194-199, 2004.
- [4] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer, "Tool support for architectural decision", in: *Proceedings of the 7th working IEEE/IFIP Conference on Software Architecture (WICSA)*, Mumbai, India, IEEE, pp. 44-53, 2007.
- [5] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions", in: *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Pittsburgh, USA, IEEE, 2005.
- [6] J. Bosch and P. Molin, "Software architecture design: evaluation and transformation", in: *Proceedings of the 6th IEEE Conference and Workshop on Engineering of Computer-Based Systems (ECBS)*, Nashville, USA, IEEE, pp. 4-10, 1999.
- [7] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. barbacci, and H. Lipson, "Attribute-based architecture styles", in: *Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSA)*, San Antonio, USA, Springer, pp. 225-243, 1999.
- [8] W. Ding, P. Liang, A. Tang, and H. van Vliet, "Knowledge-based approaches in software documentation: A systematic literature review", *Information and Software Technology*, Elsevier, vol. 56, no. 6, pp. 545-567, 2014.
- [9] D. Garlan, "Software architecture: A roadmap", in: *Proceedings of the the 22th Conference on Software Engineering, Future of Software Engineering Track (ICSE)*, Limerick, Ireland, ACM, pp. 91-101, 2000.
- [10] R. Weinreich, I. Groher, and C. Miesbauer, "An expert survey on kinds, influence factors and documentation of design decisions in practice", *Future Generation Computer Systems*, Elsevier, vol. 47, no. 6, pp. 145-160, 2015.
- [11] W. Ding, P. Liang, A. Tang, H. van Vliet, and M. Shahin, "How do open source communities document software architecture: An exploratory survey", in: *Proceedings of the 19th International Conference on Engineering of Complex Computer Systems (ICECCS)*, Tianjin, China, IEEE, pp. 136-145, 2014.
- [12] S. K. Sowe, I. Stamelos, and L. Angelis, "Understanding knowledge sharing activities in free/open source software projects: An empirical study", *Journal of Systems and Software*, Elsevier, vol. 81, no. 3, pp. 431-446, 2008.
- [13] W. Ding, P. Liang, A. Tang, and H. van Vliet, "Communicating architecture information in open source software development using mailing lists", in: *Proceedings of the 9th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Beijing, China, IEEE, 2015. (under review)
- [14] A. Brown and G. Wilson, "The Architecture of Open Source Applications", Creative Commons, 2012.
- [15] M. W. Godfrey and Q. Tu, "Evolution in open source software: A case study", in: *Proceedings of the 16th International Conference on Software Maintenance (ICSM)*, San Jose, CA, IEEE, pp. 131-142, 2000.
- [16] B. J. Williams and J. C. Carver, "Characterizing software architecture changes: A systematic review", *Information and Software Technology*, Elsevier, vol. 52, no. 1, pp. 31-51, 2010.
- [17] B. J. Williams and J. C. Carver, "Examination of the software architecture change characterization scheme using three empirical studies", *Empirical Software Engineering*, Springer, vol. 19, no. 3, pp. 419-464, 2014.
- [18] A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model", in: *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, Verona, Italy, IEEE, pp. 85-92, 2014.
- [19] J. van Gurp and J. Bosch, "Design erosion: Problems and causes", *Journal of Systems and Software*, Elsevier, vol. 61, no. 2, pp. 105-119, 2002.
- [20] C. Jensen, S. King, and V. Kuechler, "Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists", in: *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS)*, Kauai, USA, IEEE, pp. 1-10, 2011.
- [21] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology*, ACM, vol. 11, no. 3, pp. 309-346, 2002.
- [22] V. R. Basili, "Software modeling and measurement: The Goal/Question/Metric paradigm", University of Maryland at College Park, 1992.
- [23] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen, "Communication in open source software development mailing lists", in: *Proceedings of the 10th International Working Conference on Mining Software Repositories (MSR)*, San Francisco, USA, IEEE, pp. 277-286, 2013.
- [24] O. P. N. Slyngstad, J. Y. Li, R. Conradi, and M. Ali Babar, "Identifying and understanding architectural risks in software evolution: An empirical study", in: *Proceedings of the 9th International Conference of Product Focused Software Development and Process Improvement (PROFES)*, Monte Porzio Catone, Italy, Springer, pp. 400-414, 2008.
- [25] IEEE, IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software Intensive Systems, 2000.
- [26] S. A. Licorish and S. G. MacDonell, "Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study", *Information and Software Technology*, Elsevier, vol. 56, no. 12, pp. 1578-1596, 2014.
- [27] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural design decision: Existing models and tools", in: *Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA)*, Cambridge, UK, IEEE, pp. 293-296, 2009.
- [28] A. Jansen, P. Avgeriou, and J. S. van der Ven, "Enriching software architecture documentation", *Journal of Systems and Software*, Elsevier, vol. 82, no. 8, pp. 1232-1248, 2009.
- [29] N. Nurmaliani, D. Zowghi, and S. P. Williams, "Using card sorting technique to classify requirements change", in: *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE)*, Kyoto, Japan, IEEE, pp. 240-248, 2004.
- [30] M. Höst, P. Runeson, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Experimentation in Software Engineering", Springer, 2012.
- [31] J.S. van der Ven and J. Bosch, "Making the right decision: supporting architects with design decision data", in: *Proceedings of the 7th European Conference on Software Architecture (ECSA)*, Montpellier, France, Springer, pp. 176-183, 2013.
- [32] D. Pagano and W. Maalej, "How do open source communities blog?", *Empirical Software Engineering*, Springer, vol. 18, no. 6, pp. 1090-1124, 2013.

A Behavior Marker tool for measurement of the Non-Technical Skills of Software Professionals: An Empirical Investigation

Lisa L. Lacher¹, Gursimran S. Walia², Fabian Fagerholm³, Max Pagels⁴, Kendall Nygard⁵, Jürgen Münch⁶

Department of Computer Science

University of Houston-Clear Lake¹; North Dakota State University^{2,5}; University of Helsinki^{3,4,6}

Lacher@uhcl.edu¹; {gursimran.walia², Kendall.Nygard⁵}@ndsu.edu; {fabian.fagerholm³, max.pagels⁴, juergen.muench⁶}@cs.helsinki.fi

Abstract—Managers recognize that software development project teams need to be developed and guided. Although technical skills are necessary, non-technical (NT) skills are equally, if not more, necessary for project success. Currently, there are no proven tools to measure the NT skills of software developers or software development teams. Behavioral markers (observable behaviors that have positive or negative impacts on individual or team performance) are beginning to be successfully used by airline and medical industries to measure NT skill performance. The purpose of this research is to develop and validate the behavior marker system tool that can be used by different managers or coaches to measure the NT skills of software development individuals and teams. This paper presents an empirical study conducted at the Software Factory where users of the behavior marker tool rated video clips of software development teams. The initial results show that the behavior marker tool can be reliably used with minimal training.

Keywords-Non-technical Skills; behavior marker; performance.

I. INTRODUCTION

Most software is developed by teams and the success of a software project depends on the effective performance of the software project team. The PMI and the most recent PMBOK Guide [1] acknowledges that, non-technical (NT) skills in comparison to the technical skills are equally important for project success and team development. Several authors agree that the NT skills are critical to project success [2, 3]; and there are even some that assert that NT skills can have the largest impact on software development [4, 5].

The growing need for an agile workforce is one major factor that is driving the demand for NT skills [6]. Agile Manifesto's [7] first principle - "*individuals and interactions over processes and tools*" clearly points to the importance of NT skills. Agile teams depend greatly on NT skills such as efficient communication, taking responsibility, initiative, time management, and leadership.

While it is obvious that NT skills are important, and that the performance of individuals is very important to creating an effective team, there are no established guidelines for measuring team effectiveness. Different criteria for assessing team effectiveness have been identified by different authors [8, 9]. Generally, these criteria include measurements of task performance as well as the interpersonal skills of the team members. The interpersonal skills include attitudes and behaviors. Although there is extensive literature with respect to different ways to measure task performance for software development (e.g., lines of code) [10], scant research has been

performed on the measurement of NT skills, especially for software developers. A couple of notable exceptions can be found in the aviation and health care industries. Both industries have already recognized the importance of NT skills to the success of their teams, and have been using behavioral marker (BM) systems (e.g., LOSA, ANTS) to structure individual and team assessments of these NT skills. We believe that software teams can also draw upon these BM's from the aviation and health care industries. It is often Software Development managers and coaches that are responsible for assessing the performance of their development teams – not HR departments, thus a tool like a BM system needs to be available to them.

As educators and software project development managers, we are concerned with questions such as: how can managers objectively measure the NT skills of their employees to determine if their NT skills need improvement or how would feedback be provided to the team members so that they could improve their performance? This research attempts to begin answering these kinds of questions.

II. BACKGROUND –NT SKILLS, BEHAVIOR MARKERS

Non Technical (NT) Skills: NT skills are the cognitive, personal resource, and social skills that complement a person's technical skills and contribute to overall task performance [11]. Some classic examples of NT skills include communication, cooperation, decision making, leadership, stress management, and workload management. Basically; NT skills cover the cognitive and social sides of a person. In the most recent survey released by the Association of American Colleges and Universities [12], it was found that employers feel that NT skills are more important than a particular major. Several different surveys of U.S. employers have also identified a lack of NT skills as the area where young job-seekers have the largest deficiency [13]. Even professional organizations such as Professional Engineering Competence (UKSPEC), IEEE Computer Society state that professionals have an obligation to possess NT skills [14].

Behavior Markers (BM): Behavioral markers (BM) are defined [15] as "*observable, non-technical behaviors that contribute to superior or substandard performance within a work environment*". They are derived by analyzing data regarding performance that contributes to successful and unsuccessful outcomes. The overall purpose of a BM system is to use markers as a method to assess both team and individual behaviors. These BM systems provide an observation-based

method to capture and assess individual and team performance on data rather than on gut feelings. The BM tool is designed in the form of a structured list of behaviors. The Observers then use this form during a selected work situation to rate performance. This allows an individual's or team's skills to be rated in their real context. BM systems can provide a common language for giving feedback as well as discussing and teaching NT skills.

Behavior Marker (BM) Systems: BM systems have demonstrated value for assessing and providing feedback on these NT skills, for improving training programs, and in the use of building databases to identify norms and prioritize training needs. It is important to recognize that BM systems need to be specific to the domain and culture. A brief description of successful BM systems (airline, medicine) follows:

The first BM system, Line Operation Safety Audit (LOSA) is a very successful BM system that focuses on interpersonal communication, leadership, and decision making in the cockpit. Trained observers ride along in the cockpit and observe the flight crews during normal flight operations. They score the behaviors of the crew using the LOSA tool. LOSA has been endorsed by the International Civil Aviation Organization because it has been used so successfully in measuring the strengths and weaknesses of flight crews' interpersonal skills [16]. The Anesthetists' NT Skills (ANTS) [17] used in healthcare has proven very useful in assessing the NT skills of anesthetists in simulation training and has provided important performance feedback for the individuals. Another successful healthcare BM system is the Observational Teamwork Assessment of Surgery (OTAS). Many studies have shown that poor communication, coordination, and other aspects of teamwork, rather than technical failures, have been the primary causes of adverse events in surgery. OTAS has been found to be a valid measure of the NT performance of surgical teams [18].

Our goal is to develop and validate a BM system that can improve software professional team member performance by providing feedback in the form of an objective and documented assessment of the NT skills of the team members. We wanted to create a tool that is very usable by practitioners: it requires little or no training to use and does not require unreasonable effort to use. It is a concern of the researcher that if the tool took a lot of training or was too difficult to use, that the potential practitioners, such as project managers and team leads for whom the tool was meant to assist, would not find the tool useful because of the amount of effort required.

III. BEHAVIOR MARKER SYSTEM DEVELOPMENT

The development process for our behavioral marker system for software developers is detailed in our previous work [19]. As a *first* step, we performed a systematic literature review to develop NT skill inventory. The high-level question addressed by the review was: *“What are the NT skills required of software professionals performing well in their field and how can we discover what NT skills are valued by employers?”*

Details on the review protocol (sources searched, search execution, inclusion and exclusion criteria, quality assessment,

Category	Skill
Communication	Listening
	Oral Communication
	Persuasion
	Questioning
Interpersonal	Written Communication
	Ability to receive criticism
	Assertiveness
	Attitude
	Culturally sensitive
	Diplomacy
	Information Sharing
	Interpersonal Relationships
	Leadership
	Negotiation
	Patience
Politics	
Problem Solving	Self-Esteem
	Social Sensitivity
	Teamwork
	Attention to Details
	Critical Thinking
	Judgment and Decision Making
	Learning
	Problem Sensitivity/Contextuality
	Problem Solving
	Visualization
Flexibility	
Work Ethic	Initiative/Motivation to Work
	Organized
	Professionalism
	Responsibility
	Stress Tolerance
Time Management	

Fig. 1: Desired NT skills of Software Professionals

data extraction) can be referred to in a report [20]. The output of this step was an initial list of 35 NT skills that were clustered into four major categories: communication, interpersonal, problem solving, and work ethic (see Fig. 1). The detailed description of each skill can be referred [20].

During the *second* step, the initial list of NT skills had their quality assessed and were validated by focus group of experts in industry and academia. Two surveys (and focus groups) were conducted online (using a cross sectional design) to gather NT skill priorities, missing NT skills, description clarifications, and examples of good and poor behaviors for the top rated NT skills of software developers. So that we could prioritize our efforts, focus group ranked the importance of each NT skill to software professionals during the first survey. After the survey analysis, we had a reduced list of 16 skills to focus on. During the second focus group survey, we gather a total of 408 examples of observable actions that indicated good performance and behavior of each NT skill as well as examples of observable actions that indicate poor performance and behavior of each NT skill. These examples were reviewed, clarified, and redundancies were eliminated. The final set of NT skills consisted of: *teamwork, initiative/motivation to work, listening, attitude, critical thinking, oral communication, problem solving, attention to detail, flexibility, integrity/honesty/ethics, time management, and questioning.* Some behavioral examples, such as *“being a good team player”* and *“body language and persona emitting that you do not enjoy your work”*, were too ambiguous and removed. It was also felt that the *“Leadership”* skill did not have enough observable

Listening	
Paying attention to and concentrating on what is being said, repeating points of discussions to ensure mutual understanding and asking questions that refine points about which one is uncertain.	
Examples of Good Behaviors	Examples of Bad Behaviors
Restates, rephrase (paraphrase), or summarize the message to provide feedback if the message was clear and understood. Questions of confirm understanding of the message (e.g. Listening carefully to a warehouse presentation, asking questions, and providing constructive feedback in a supportive way).	Passively participating and not paying full attention. (e.g. Looking around the room, checking emails, or other activities on a laptop or phone that show they are not paying full attention to the speaker).
Pauses before restatement or question to show that the message was carefully considered.	Creates distractions while someone else is talking such as whispering to others while someone else is speaking.
Asks clarifying questions. E.g. asks questions to ensure design met the requirements.	Cutting in to the conversation, not letting others complete thoughts. Talking at the same time as someone else.
Lets others talk; does not interrupt.	Combative listening mode - not receptive to other points of view and immediately want to promote their point of view.
Looks at the person speaking. Maintains good eye contact with the speaker.	Exhibits poor body language: crossing arms and legs, stands too close or too far away to the speaker and causes the speaker noticeable discomfort.
Exhibits receptive body language such as leaning forward, nodding or shaking head, etc. (note that nonverbal cues vary from culture to culture. e.g. in some cultures vertically nodding head usually means agreement, in other cultures shaking head means agreement)	

Fig. 2: Example of “Listening” behaviors (good and bad examples)

behaviors that would be able to be clearly identified, so that NT skill was removed. The result of the second survey was a behavior-based software engineer NT skills taxonomy. Fig. 2 shows the resultant examples of good and poor behavior for the “Listening” skill. The same process was used to create examples of good and poor behavior for each NT skill.

During the *third* step, the behavior marker systems being used in aviation, health care, rail transport and maritime transport were examined. Each system’s structure was examined to select which elements would have the most potential for use in software development and our final tool was a composition of several systems. The NT skills validated by the focus group along with the good and bad behavior examples for those skills were structured into a BM audit tool for software development. For reference, we refer to the BM audit tool as the Non-Technical Skill Assessment for Software Developers (NTSA).

The NTSA is designed to be used by an observer (i.e. manager, team leader, coach) during routine team interactions or meetings. It is intended that each time a behavior is observed, a mark is placed in the appropriate column by placing a tick mark in that column: observed and good, or expected but not observed. Observations can be clarified by placing explanations in the comments section. The observer can see skill definitions and examples of good and poor behavior for a particular behavioral marker by viewing the second page. A manager is allowed to list as many or as few skills as desired in the behavioral marker column. The observer will score the behaviors based on how well the behavior meets the behavioral examples and its definition.

IV. EMPIRICAL VALIDATION OF BEHAVIOR MARKER

In order to evaluate our BM tool, an empirical study rated video clips of student software development teams that were working on industrial strength projects within the Software Factory (as shown in Fig. 3 and explained).

1) Software Factory Background

The Software Factory is a software development laboratory created by the University of Helsinki, Department of Computer Science. All research was performed in Finland due to the requirements of international privacy laws. The University of Helsinki is consistently ranked in the top 100 out of world’s 15,000 universities, in part because the university promotes science and research together with European’s top research-intensive universities. The master’s degree programs are taught in English in order to support the large number of international

students who study at the university. The Software Factory’s primary participants are students, but the businesses provide team members who work with the students, and university faculties oversee the projects, although the faculty involvement is kept to a minimum. Almost all project communication is in English. Faculty involvement consists primarily of project orientation and project intervention if problems cannot be resolved by the students, coach, and customer. The coach is generally an upper level student with Software Factory project experience. University students take on the role of the development team for projects provided by businesses. The customer has company representatives that take on the role of the product owner and represents the interests of the company. Although these representatives are not co-located, they do come by the Software Factory for weekly demos, sometimes for meetings, and are generally available via telephone and email. Researchers are able to observe what happens in the project due to the seven cameras that provide multiple angles of view and four microphones that record activities in the Factory room. In Software Factory projects, the participants take on the core roles of a typical Scrum project. Projects at the Software Factory last for seven to eight weeks; the students work approximately 6 hours per day, 4-5 days per week.

2) Study Design

This study investigates whether the BM system can be used with consistency by different raters to capture a measurement of the NT skills of software developers, thus facilitating objective feedback to software development teams and individuals. This study used a blocked subject-project study. This type of analysis allows the examination of several factors within the framework of one study. Each of the non-technical skills to be studied can be applied to a set of projects by several subjects and each subject applies each of the non-technical skills under study. In this study, raters evaluated the NT skills of project teams using the NTSA tool. The project teams worked together using state-of-the-art tools, modern processes and best practices to prototype and develop software for real business customers in an environment that emulates industry. Video tapes of the projects were evaluated to rate the student team’s NT skill performance. The details of the study are provided as follows.

Independent and dependent variables: The experiment manipulated the following independent variable:

- a) Behavioral Marker System tool and Example Behaviors: Each non-technical skill has its own set of good and



Fig. 3: Software Factory

poor behavioral examples that are used by the raters to evaluate team performance of each non-technical skill.

The following dependent variable was measured:

b) Rater’s Evaluations: The behavioral rating for each non-technical skill by each rater. This measure includes the percent positive for each rater for each non-technical skill.

Participating Subjects: The participant subjects (students in the Computer Science master’s degree) were software developers from two different projects. There were two different projects that were evaluated. One project had five team members and the other had seven team members. The students worked together to develop a software solution to a project posed by the business customer.

Artifacts: Although the NTSA tool could be used to evaluate the NT skills of both individuals and teams, it was decided to test for team skills first. Because we were primarily interested in how the team member’s NT skills manifested when interacting with others, it was decided that the first clips to be evaluated would be of team meetings, and so standup meetings, impromptu team meetings, and customer demos were targeted. After extracting all of these clips, it was determined that we would focus on standup meetings because of the consistency and quantity of footage. Two raters used the NTSA tool to independently rate each clip. The NTSA was in the form of a spreadsheet on a computer.

Experiment Procedure: Study steps as described below:

Step 1 – Project Selection: We decided to focus on two projects. We selected one project that had gone well and one that had not gone well (as the first project) in the expectation of producing diverse scorings.

Step 2– Video Clip Collection: Video and audio recordings of the entirety of each project were collected. The Software Factory deployed 7 video cameras and 4 microphones. The cameras were situated such that one could not actually view what was on the computer monitors or clearly see any of the paper artifacts, although anything written on the white board or displayed on either of the two projectors could be clearly viewed. Video clips were labeled with the type of meeting along with date and start and end times so if the clip became corrupted and needed to be re-created, the researcher would know exactly what day and time to go retrieve the clip. A spreadsheet was used to store this information along with which cameras and microphone were used in the clip.

Step 3 – Test Rater Understanding of the NT Skill and Behavioral Descriptions: During the initial phase of the empirical evaluation, two researchers from the Software Factory reviewed the NTSA tool to make sure they understood the descriptions of the good and poor behaviors. Each researcher has extensive experience with project teams in the Software Factory. Each of the researchers reviewed the behavioral descriptions independently, and added comments. Then we met as a group to discuss potential changes. Following the discussion, some behavioral descriptions were modified, some eliminated and some added. Ultimately, the group reached a consensus on all descriptions. It was also determined that it

was unrealistic to observe the behaviors for Integrity, Honesty, and Ethics, Attention to Detail, and Time Management and that it would be better to look at other documents and devices, such as Kanban metrics, bug reports and customer feedback to observe and rate those non-technical skills.

Step 4 – Test Usability of the Tool: The Software Factory researchers used the initial NTSA tool to evaluate several clips to test usability. First, each researcher reviewed the descriptions of each behavior and the good and poor behavioral examples. Then, each researcher did independent evaluations of the clips, after which we met for discussion of the evaluations. There was consensual agreement that fine gradations in quality were difficult to determine and the researchers agreed that the tool would only include ratings for good and poor behavioral observations. The final NTSA tool is shown in Fig. 4. The raters also noted that it was very difficult to determine how often to place a mark for exhibition of good and poor behavior because the meetings were continuous. Because the raters are not classifying discreet events or statements, it was decided that the raters would be notified when a minute had passed, which would prompt them to decide if the team exhibited any good behaviors or poor behaviors and to put a mark in the appropriate column. If they did not feel that any good or poor behaviors were exhibited by the team, they did not place a check mark. If they felt that both good and poor behaviors were exhibited, they put a check mark in each column. After the evaluation of the last clip and post discussion, there was consensus that the tool was ready for testing.

Step 5 – Actualizing Rater’s Evaluations: Each rater individually rated forty five standup meetings over the course of ten weeks. The time spread of the ratings simulates the frequency with which a manager, team lead, or coach would use the tool. We also wanted to eliminate the amount of fatigue that could transpire. The raters used the spreadsheet version of the NTSA behavioral marker system tool with the one minute timer. Unlike the trial evaluations, the raters rated all NT skills while viewing the video clip as opposed to only rating one non-technical skill per viewing.

Non-Technical Skills (NTSA) Assessment Instrument, Final Version				
Date: _____		Observer: _____		Segment: _____
Event (place check mark): Stand up Meeting: ___ Customer Demo: ___ Other (describe): _____				
Category	Skill	Good	Poor	Comments
Communication	Listening			
	Oral Communication			
	Questioning			
Interpersonal	Attitude			
	Teamwork			
Problem Solving	Critical Thinking			
	Problem Solving			
Work Ethic	Flexibility			
	Initiative/Motivation to Work			

* Place one mark in the appropriate column each time behavior is observed.

Fig. 4: NT skills assessment instrument

V. RESEARCH RESULTS

Because we were primarily interested in how the team member's NT skills were manifested when interacting with others, it was decided that standup meetings would be the focus of our analysis. We were able to limit the video footage to view based on the schedule that the development team agreed upon. Generally, the team limited their development efforts to Monday through Friday from eight in the morning to five in the afternoon. Thus, for a typical seven to eight week time period, this means that there were approximately 2,205 to 2,520 hours of video footage per project available, with four different audio choices for each hour.

We evaluated the percentage of positive ratings, and developed a binary data set for statistical analyses. By inspecting the distributions of the raters when examining the skills, a critical value (specific to each NT skill) was chosen to separate the 0 or 1. For example, for the Listening NT skill, a critical value of 0.8 was chosen. This value was chosen because it approximately separated the raw data evenly into two parts. Thus, if the good percentage was greater than or equal to 0.8, the rating was assigned to 1, and the rating was assigned to 0 if the good percentage was less than 0.8. Using this information, a 2X2 table containing the good and bad percentages of two raters was created. Next, a McNemar's test was used to evaluate whether or not there are significant differences between the raters. A value of $p < 0.05$ would tell us that there is a significant difference between the raters and p value greater than 0.05 would signify inter-rater reliability.

As mentioned earlier, an analysis of the quantitative data includes the rater's evaluations for good and poor behaviors observed in the standup meetings. It was decided to follow John Uebersax's [21] recommendation to run McNemar's test of marginal homogeneity and calculate the inter-rater reliability between two individuals. Cohen's kappa could not be used because the sample size was not large enough to be reliable.

To analyze the agreement between the two raters, analyses were performed for each of the nine NT skills: *listening, oral communication, questioning, attitude, teamwork, critical thinking, problem solving, flexibility, and initiative and motivation to work*.

thinking, problem solving, flexibility, and initiative and motivation to work. Figure 2 shows the McNemar test results for each of the NT behaviors evaluated.

To test this study hypothesis, we ran McNemar's on the percentage positive ratings (calculated to produce a binary data set) for each rater and for each NT skill to test for rater agreement in cases where there were enough observation data points. The results showed that, inter-rater reliability of NTSA was found for eight of the nine NT skills in the tool. These results provide initial evidence that NTSA can be a useful tool that could be easily used by managers, team leaders, etc. responsible for the development of these skills, to objectively and consistently measure their employee's NT skills. A tool, such as the NTSA, provides a mechanism to not only improve a team and by extension the software that they produce.

The fundamental finding is that inter-rater reliability of NTSA was found for eight of the nine NT skills in the tool. The "Problem solving" NT skill needs further enhancements and subsequent validation before it could be used. In fact, it is possible that "problem solving" simply is not observable. The Non-Technical Skills Assessment for Software Developers (NTSA) system can be used reliably by individuals responsible for the NT skills of software development teams, such as educators, managers, team leads, etc. Although the raters did practice rating several video clips with the tool, and this is equivalent to a few meetings, it is also very interesting to note that the raters do not need to be human factors experts, nor did it require extensive initial training for the tool to be used reliably. Although the raters felt that it was very easy to use the tool in its spreadsheet form while working with the form on a computer where the behavioral examples are only a click away, they also noted that they would like to keep the electronic capability if they were rating a live event rather than a video recorded event. The raters also noted that the tool could be customized to only include the NT skills of interest to the rater – not all non-technical skills need to be rated at the same time. This would make the tool even easier to work with. While, these results are encouraging, only two projects and two raters were used. Therefore, more studies need to be performed. A positive aspect of this study is that the raters had different levels of project management experience, and were able to use to tool and get reliable results.

VI. THREAT TO VALIDITY

Although the results of this study are encouraging, there are certain threats to validity that exist. One such threat is that only two projects were evaluated. Like any study, the more a subject is tested, the more empirical studies that are performed, the more one can see if the results are repeatable. Rater agreement testing should continue to be performed on more projects. Another threat is that both projects were rated by the same two judges. More empirical will be performed with different raters using the NTSA tool to ensure the robustness of the tool. One positive aspect about the raters is that each had different levels of software development project management experience. That



Fig. 5 Aggregation of McNemar Test Results

means that the raters do not have to have the same level of experience or backgrounds in order to use the tool and get reliable results. Another potential threat is that both projects were fairly successful, and thus may not have exercised the poor behavior examples enough. Lastly, the projects were performed by student teams and thus many not be generalizable; although this threat was mitigated by the level of professional business-like environment that can be found in the Software Factory and by the fact that both projects were real-world projects.

VII. CONCLUSION AND FUTURE WORK

Our results establish that the NTSA tool can be reliably used with minimal effort. This is valuable knowledge for managers and educators. We recognize that teams need members with the correct technical skill set and knowledge, by using NTSA software development team managers can identify the areas in which the team's NT skills could use some improvements. Using the same tool on subsequent projects will allow us to determine if there was any improvement in a given skill. Such as tool provides a mechanism with which to improve a team and by extension the software they produce. The NTSA provides a common language with which to understand and communicate about NT skills important to software professionals

In the future, we would plan on repeating this study on other projects. Specifically, we would like to use the tool on more unsuccessful software development project to see if there is a correlation between poor NT skills and an unsuccessful project. This research can be extended to include all of the NT skills deemed important to software developers as identified in the NT skills taxonomy. This would give educators and managers a rich set of NT skills and behaviors that could be evaluated. This tool also needs to be tested on individual software developers within software development teams to see if it can be effectively used to assess the NT skills of the individual as well as the team. This tool should also be tested in industry to verify that it works for professional software developer and teams, as well as student software development teams.

REFERENCES

[1] Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). Newton Square, PA: Project Management Institute, 2008, pp. 215.

[2] S. Acuna, N. Juristo, and A.M. Moreno, "Emphasizing Human Capabilities in Software Development", IEEE Software, vol. 23, 2006, pp. 94-101.

[3] E. Amengual, and A. Mas, "Software Process Improvement through Teamwork Management," in Proceedings of the 8th International Conference on Product-Focused Software Process Improvement, 2007, pp. 108-117.

[4] A. Cockburn, and J. Highsmith, "Agile software development: The people factor", Computer, vol. 34, 2001, pp. 131-133.

[5] N. Gorla, and Y. Wah Lam, "Who Should Work With Whom?" Communications of the ACM, vol. 47 No. 6, pp. 79-82, Jun. 2004.

[6] Abell, Angela, Information World Review; Dec 2002; 186; ABI/INFORM Complete pg. 56.

[7] <http://agilemanifesto.org/>

[8] S.G. Cohen, and D.E. Bailey, "What Makes Teams Work: Group Effectiveness Research from the Shop Floor to the Executive Suite", Journal of Management, vol. 23, 1997, pp. 239-290.

[9] J.J. Jiang, J. Motwani, and S.T. Margulis, "IS team projects: IS professionals rate six criteria for assessing effectiveness", Team Performance Management, vol. 3, 1997, pp. 236-242.

[10] O. Hazzan and I. Hadar, "Why and how can human-related measures support software development processes?" The Journal of Systems and Software 81, 2008. Pp/ 1248-1252.

[11] R. Flin, P. O'Connor, and M. Crichton., "Safety at the sharp end: A guide to non-technical skills", 2008, Burlington, VT: Ashgate Publishing Company. Pg. 264

[12] Higher Ed News, "Survey Finds Business Executives Aren't Focused on Majors They Hire" accessed Mar. 14, 2014,

[13] <http://business.time.com/2013/11/10/the-real-reason-new-college-grads-cant-get-hired/>

[14] UKSPEC,"UK-SPEC UK Standard for Professional Engineering Competence," accessed Mar. 14, 2014, [www.engc.org.uk/ecukdocuments/internet/document library/UK-SPEC third edition.pdf](http://www.engc.org.uk/ecukdocuments/internet/document%20library/UK-SPEC%20third%20edition.pdf)

[15] B. F. Klampfer, R. L. Helmreich, B. Hausler, B. Sexton, G. Fletcher, P. Field, S. Staender, K. Lauche, P. Dieckmann, and A. Amacher. "Enhancing performance in high risk environments: Recommendations for the use of behavioral markers." Behavioral Markers Workshop, 2001, pp. 10.

[16] B. F. Klampfer, R. L. Helmreich, B. Hausler, B. Sexton, G. Fletcher, P. Field, S. Staender, K. Lauche, P. Dieckmann, and A. Amacher. "Enhancing performance in high risk environments: Recommendations for the use of behavioral markers." Behavioral Markers Workshop, 2001, pp. 10.

[17] G. Fletcher, R. Flin, P. McGeorge, R. Glavin, N. Maran and R. Patey, "Development of a Prototype Behavioural marker System for Anaesthetists' Non-Technical Skills (ANTS)," Workpackage 5 Report, Version 1.1. (2003)

[18] G. Fletcher, R. Flin, P. McGeorge, R. Glavin, N. Maran and R. Patey, "Development of a Prototype Behavioural marker System for Anaesthetists' Non-Technical Skills (ANTS)," Workpackage 5 Report, Version 1.1. (2003)

[19] L.L. Bender, G.S. Walia, F. Fagerholm, M. Pagels, K.E. Nygard, and J. Münch, "Measurement of Non-Technical Skills of Software Professionals: An Empirical Investigation", Proceedings of the 26th IEEE International Conference on Software Engineering and Knowledge Engineering. July 1- 3, SEKE 2014 Vancouver, Canada.

[20] L.L. Bender and G.S. Walia, "Measurement of Non-Technical Skills of Software Development Teams", Department of Computer Science, North Dakota State University, Fargo, ND, Tech. Rep. NDSU-CS-TR-14-001, Mar. 2014.

[21] J. Uebersax. "Statistical Methods for Rater and Diagnostic Agreement" Internet: <http://www.john-uebersax.com/stat/agree.htm> [Apr. 14, 2013]

A Platform for Empirical Research on Information System Evolution*

Robert Heinrich¹, Stefan Gärtner², Tom-Michael Hesse³, Thomas Ruhroth⁴,
Ralf Reussner¹, Kurt Schneider², Barbara Paech³, and Jan Jürjens⁴

¹Karlsruhe Institute of Technology, Germany, {heinrich, reussner}@kit.edu

²Leibniz Universität Hannover, Germany, {stefan.gaertner, kurt.schneider}@inf.uni-hannover.de

³University of Heidelberg, Germany, {hesse, paech}@informatik.uni-heidelberg.de

⁴TU Dortmund, Germany, {thomas.ruhroth, jan.jurjens}@cs.tu-dortmund.de

Abstract

Software-intensive systems are subject to continuous change due to modification of the systems themselves and their environment. Methods for supporting evolution are a competitive edge in software engineering as software is operated over decades. Empirical research is useful to validate the effectiveness of these methods. However, empirical studies on software evolution are rarely comprehensive and hardly replicable. Collaboration in empirical studies may prevent these shortcomings. We analyzed the support for such collaboration and examined existing studies in a literature review. Based on our findings, we designed CoCoMEP – a platform for supporting collaboration in empirical research on software evolution by shared knowledge. We report lessons learned from the application of the platform in a large research programme.

1 Introduction

In industrial practice, many information systems [1] are operated over decades. During operation they face various modifications, e.g. due to emerging requirements, bug fixes, and environmental changes, such as legal constraint or technology stack updates. In consequence, the systems change continually which is named software evolution [2]. Supporting software evolution is a competitive advantage in software engineering. A variety of methods aim at supporting different aspects of software evolution. However, it is hard to assess their effectiveness and to compare them due to divergent characteristics. Empirical research in terms of case studies and controlled experiments is useful to validate these methods. However, empirical studies on software evolution are rarely comprehensive. They often cover only one of the many aspects needed to study evolution:

(i) long time-frames of observation are required to analyze changes, (ii) large amount of artifacts and (iii) various types of artifacts are affected by evolution, (iv) artifacts repeatedly change, (v) changes partly build upon each other, (vi) various stakeholders are involved, (vii) access to relevant project data, (viii) relevant project data must be documented over long time spans, (ix) relevant context knowledge must be documented beyond the code base and issue trackers.

To study evolution comprehensively, we believe it is important to collaborate by joint research in order to increase coverage of the aspects. Joint research supports sharing of knowledge and resources [3]. In particular, this allows replicating studies which in general is important to confirm and to strengthen results of empirical research [4] and thus enhance evidence. Our goal is to support joint research by collaboration and replication in empirical studies based on common evolution scenarios and artifacts. Currently, empirical studies on software evolution are seldom comparable as they vary in analyzed subjects and execution process. Furthermore, these studies are rarely reusable as important artifacts (e.g., requirements, design decisions, or context knowledge) are often not provided to the community. To the best of our knowledge, there is neither a community-accepted case study for software evolution nor a common benchmark available. Consequently, a common basis for study collaboration and replication is missing.

In this paper, we propose CoCoMEP¹ – a platform for collaborative empirical research on information system evolution. Under a “platform” we understand a comprehensive knowledge base for the evaluation process that can be exploited and extended by other researchers with different backgrounds and research interests. It provides assistance on diverse characteristics important for software evolution, e.g. the life-cycle of the system, artifacts in different revisions, and comprehensive evolution scenarios.

*This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

¹The term is a combination of Common Component Modeling Example “CoCoME” [5] and “Platform”

CoCoMEP builds upon the established CoCoME case study [5]. CoCoMEP is already in use for collaboration between several projects within the DFG Priority Programme *Design For Future - Managed Software Evolution* (SPP1593) [6]. These projects collected knowledge on experiences and lessons learned on research collaboration in software evolution. CoCoMEP, however, is not limited to SPP1593 but open for reuse and extension by researchers outside the scope of the priority programme. For constructing CoCoMEP, we first analyzed the current support for research collaboration (Sec. 2). Second, we conducted a literature review to examine existing empirical studies (Sec. 3). Based on identified issues and requirements derived, we designed CoCoMEP (Sec. 4). We discuss lessons learned from applying CoCoMEP in SPP1593 (Sec. 5). The paper concludes in Sec. 6.

2 Related Work in Empirical Research

In this section, we analyze related work with regard to collaboration. In particular, we focus on replicability and comparability that are both indispensable to enable research collaboration. The aim is to learn from experiences in empirical research and derive requirements (**R1-6**) as basis for the design of CoCoMEP. On this account, we focus on how other research communities standardized their evaluations to compare different solutions. We are interested in properties that enable or constrain comparability. Furthermore, we examine papers discussing replication in empirical research to consider replicability in our platform.

Standardized Evaluation within certain Research Communities: In Espinha et al. [7], a standard and open-source case study for SOA is proposed. The authors discussed that for case studies in the SOA research community a wide variety of small and closed systems is used limiting comparability of obtained results. Therefore, standardized study subjects are needed to enable assessment of ideas and methods within this community. Another attempt to standardize evaluation has been made in Proksch et al. [8]. They described a framework focusing on evaluations of developer assistance tools. As discussed in the mentioned papers, the standardized case studies have to address the major challenges within a particular community. Otherwise, it will not find broad acceptance.

To compose suitable case studies, we can learn a lot from the repository mining community. In this community, development histories (repositories) are analyzed with respect to certain research questions. To compare results, some projects (e.g., Apache Tomcat, Mozilla Firefox, etc.) exist that are used by several research teams. For example, Lott et al. [9] and Lessmann et al. [10] proposed frameworks for comparative software defect prediction experiments. According to this, all artifacts of the study subject should be

made available to motivate a certain community to conduct necessary empirical studies and to achieve better and more convincing result as well as research collaboration. Hence, we consider standardized evaluation as requirement **R1** for joint empirical research.

Replication of Empirical Studies: The aim of standardized evaluation is to enable comparison and replication of empirical and complementary studies. As described in Juristo et al. [4], replication in empirical studies is required to confirm or deny original results as well as to complement the original experiment. However, experiences have shown that replication is hard to achieve [4]. One reason is to establish identical conditions of the experimental context which might be impossible in some cases. In addition, the replicating researchers need to fully understand the experimental design, but most rationale is not provided (tacit knowledge). To enable replication effectively, Schull et al. [11] proposed to provide laboratory packages for experiments. The authors defined a laboratory package as an experimental infrastructure including experiment design, necessary material, and possible variation points. As stated in Mendonça et al. [12], the problem is to compose static laboratory packages that cover all aspects of the experiment. The replicating researchers need to fully understand the experiment and the corresponding material to avoid unpredictable variants in the experiment limiting the meaning of the achieved results. As a consequence, Mendonça et al. proposed that effective replications also require well-defined processes involving the original researchers. This implies the need for an effective collaboration structure among researchers, which we consider as requirement **R2**. For this purpose, they introduced a framework for improving the replication of experiments (FIRE) and emphasize the importance of knowledge sharing for internal and external replications (e.g., experimental details and rationale).

Requirements on a Research Platform: As stated in Demeyer et al. [13], case studies are popular for assessing new approaches relating to evolution, but most of them use toy examples that have a bias towards the approach. Moreover, as stated in Runeson et al. [14], different study subjects and missing documentation of the evaluation process decrease replication and comparability of case studies. To cope with these issues, a standardized study as well as evaluation process is needed. Regarding evolution, Demeyer et al. proposed a set of requirements. **R3:** the case must comprise artifacts that correspond to all life-cycle phases (life-cycle requirement). **R4:** the evolution process must contain iterations and increments (evolution requirement). **R5:** the application, problem, and solution domains of the case must be qualified (domain requirement). **R6:** tools necessary to replicate the case must be evaluated (tool requirement). Our research platform must address these six requirements properly.

3 A Literature Review on Empirical Studies

To understand how well existing studies on software evolution support the requirements identified above we conducted a literature review. This section provides the search strategy, process documentation, and findings of the literature review. Basically, the review followed the guidelines by Kitchenham and Charters [15]. However, it was not conducted as a strict review as every paper was only reviewed by one of the authors. As empirical methods we considered case studies and experiments. We neither aimed at giving a comprehensive overview of approaches for supporting software evolution nor at presenting the approaches found. The research questions (RQ) for our review were:

Which aspects of evolution are addressed explicitly in case studies and experiments? (RQ1) According to the requirements of Demeyer et al. [13], we consider the following aspects of evolution: *Life-cycle* which covers the artifacts, activities and their relationships that correspond to all phases in the system's life-cycle. *Evolution process* covering iterations in the life-cycle which we surveyed by the time horizon of the study (i.e. design-time, run-time, or post-mortem). *Domain* which covers the artifacts to provide a concrete study setting. We did not include the requirement *tool* as this is hard to examine by literature review.

With **RQ2** we again refer to the focus of Sec. 2 by asking **how is comparability and replication supported in case studies and experiments?**

3.1 Paper Search and Selection Process

To answer the research questions publications were required to be related to *evolution*, *information systems* or *software engineering*, and *empirical studies*. In consequence, three sets of keywords were created. The evolution set covers the keywords *evolution*, *maintenance*, *changeability*, and *modifiability*. The domain set contains the keywords *information system* and *software*. The methods set comprises *case study* and *experiment*. We did not derive search terms from RQ2 as comparability and replicability is typically not stated within the single study, but has to be assessed manually. We searched journals (e.g., ESEJ, TOSEM, TSE, KAIS, and ICSM) and conference proceedings (e.g., CSMR, ESEM, ICSE, and FSE) related to empirical research and software evolution with an impact factor greater or equal one and an acceptance rate lower than 30%, respectively.

We performed two selection iterations on the initial amount of 272 search hits. Each iteration was performed by one author guided by defined inclusion and exclusion criteria as proposed by Kitchenham and Charters [15]. The first iteration evaluated whether the papers conformed to the formal requirements on case studies and experiments

as described by Runeson et al. [14]. In the second iteration, the contribution of the papers to one or both research questions was evaluated. After the first iteration 105 papers were selected for further analysis. Within the second iteration 53 papers were identified that contribute to the research questions. The identified papers are listed online (www.dfg-spp1593.de/cocome/platform) due to page restrictions in this paper.

3.2 Findings

As a general answer to RQ1, no study has been found considering the entire evolution life-cycle. In addition, neither artifacts nor relations between the different development activities are comprehensively covered by existing studies. Distributions for the findings are depicted in Fig. 1. *Focus on design-time*: Our review shows that design-time and post-mortem studies (52 out of 53) outweigh run-time studies (1 out of 53). *Focus on a specific activity*: Most studies are only focused on a specific activity within the life-cycle. In particular, requirements engineering and maintenance phase were least covered. *Focus on a specific artifact*: For supported activities, the approaches usually consider a typical type of artifact, like code or UML models. Only a few studies (2 out of 53) focus on changes of additional documentation. We could not find a study focusing on changing decisions during evolution. Moreover, only a few studies (10 out of 53) cover co-evolution of development artifacts. The majority of these (6 out of 10) covers co-evolution within the same type of artifact, like co-evolution of components or test cases. *Relationships between activities mostly not considered*: Only a few studies (8 out of 53) examine the relationships between activities within different life-cycle phases (requirements engineering and implementation, design and implementation, implementation and maintenance).

The following findings answer RQ2. *Missing comparability and replicability of studies*: Most empirical studies and their results are not *comparable* in terms of domain, size, or complexity. In particular, this is true for controlled experiments, where the complexity of tasks is limited which may lead to less realistic settings. Thus, the obtained results have only limited evidence for software evolution in practice. This is related to the problem of *replicating* empirical studies [4]. Regarding software evolution, replication is difficult to achieve due to a large amount of changes required in the study subjects within a long period of time. Consequently, a complete change history would be required for the study subjects. Moreover, no study in our review made a clear distinction which types of evolution were addressed by given changes. As introduced by Lientz and Swanson [16], three types can be distinguished – *corrective*, *perfective* and *adaptive* evolution. If a case study does not specify the ad-

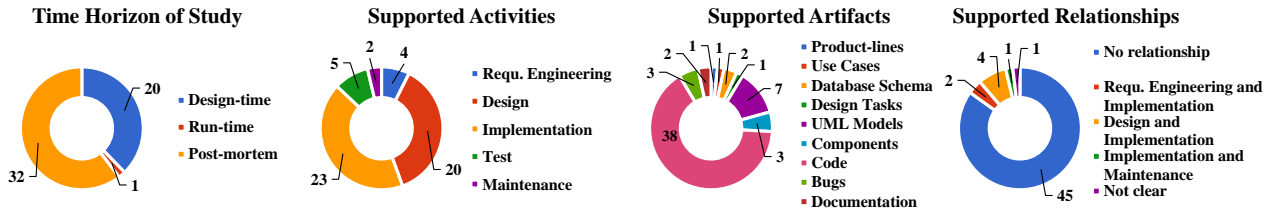


Figure 1. Distribution of Supported Development Time Horizon, Activities, Artifacts and Relationships

addressed evolution type, it is difficult for other researchers to assess whether the study is appropriate for their approach.

Only a few publications provide enough details about their empirical study to enable replication. Most of these studies are performing a post-mortem analysis on code repositories. However, there exist only a few open-source projects for repository mining studies, on which the community for post-mortem analysis agreed. Overall, no common guidelines have been found for studies on software evolution in order to support joint research.

4 The CoCoME Platform

The findings of our literature review clarified the need for improvement in case study research on information system evolution. According to the requirements identified in Sec. 2, we developed the research platform CoCoMEP depicted in Fig. 2. On this account, the established CoCoME system [5] serves as the study subject (Sec. 4.1). We developed examples of change scenarios in information system evolution (Sec. 4.2), constructed sample activities in system development and operation, and arranged them in life-cycle form (Sec. 4.3).

4.1 Evolution Subject

An evolution subject is the amount of artifacts in different revisions (e.g., requirements or monitoring data) that represent an information system. We used CoCoME [5] as evolution subject. CoCoME has been set up in a Dagstuhl research seminar as a common case study on which several methods in the context of component-based software engineering have been applied. Since more and more people do research on software evolution, CoCoME has been applied in new areas as a demonstrator for software evolution methods. CoCoME represents a trading system as

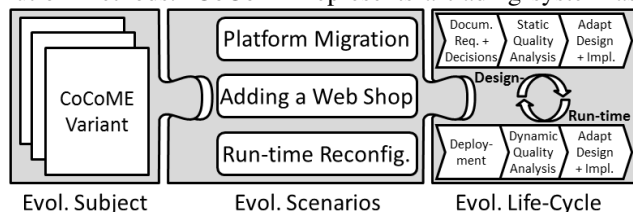


Figure 2. Overview of the CoCoME Platform

it can be observed in a supermarket chain handling sales. This includes processing sales at a single store of the chain, e.g. scanning products or paying, as well as enterprise-wide administrative tasks, e.g. inventory management or reporting. A detailed description of CoCoME is given in [5]. Since CoCoME has been applied and evolved successfully in various research projects, e.g. SLA@SOI (<http://sla-at-soi.eu>) and Q-Impress (www.q-impress.eu), several variants exist that span different platforms and technologies, such as plain Java code or service-oriented frameworks. Furthermore, various development artifacts are available, such as requirements specification or design documentation, which changed over time. CoCoME is well suited to serve as evolution subject because the supermarket context is commonly comprehensible and the complexity of the system is appropriate. As CoCoME is a distributed system, several quality properties are affected by evolution.

4.2 Evolution Scenarios

An evolution scenario describes changes to a certain evolution subject. Based on CoCoME, we implemented distinct evolution scenarios (S1-S3) covering the categories adaptive and perfective evolution (cf. Sec. 3). Corrective evolution is not considered as this merely refers to fixing design or implementation issues. A perfective evolution with regard to a changing environment is represented in S1 by emerging user requirements. An adaptive evolution is reflected in S2 by platform alterations due to evolving technology. Furthermore, in order to accommodate the self-adaptiveness of modern software architectures, reconfiguration during system operation is addressed in S3. Implementation details are visualized online (www.dfg-spp1593.de/cocome/platform) due to page restrictions.

S1: Web Shop Extension: A web shop is added where the customers can order online and pick-up the goods at the store. This design-time modification includes adding new use cases and modifying existing design models. S1 represents a requirements-driven evolution that transforms a closed system (only employees can access) to an open system (customers can accessed via internet). Hence, various quality properties are affected, e.g. privacy, security, performance, and reliability.

S2: Platform Migration: The enterprise server and its connected database are now running in the Cloud to reduce

operating costs of resources. The introduction of the Cloud enables flexible adaptation and reconfiguration of the system, however, causes new challenges regarding aforementioned quality properties.

S3: Database Migration: During a big advertise campaign, the performance of the system may suffer due to limited capacities of the Cloud provider currently hosting the database. Migrating the database from one Cloud provider to another may solve the scalability issues. S3 represents a reconfiguration at run-time. Migrating the database may cause privacy issues, as described in further detail in [17].

4.3 Evolution Life-Cycle

An evolution life-cycle integrates activities and their relationships required to implement one or more evolution scenarios. We developed a set of sample activities typical in information system evolution and arranged them in life-cycle form (cf. Fig. 2) to cope with aforementioned evolution scenarios.

An iteration in the life-cycle starts with a change request, e.g. for S1 or S2. Decisions are made and documented. A static quality analysis is conducted to identify quality issues at design-time. The design is adapted and implemented. After deployment, a dynamic quality analysis is conducted for the running system which may result in automated adaptation at run-time (S3) or a new iteration for manual evolution.

The life-cycle addresses the findings from literature review as it (i) spans design-time and run-time, (ii) covers various activities located in different phases of software development and operation, (iii) contains a variety of heterogeneous artifacts associated to the life-cycle activities, e.g. requirements, decisions, UML models, monitoring data, simulation data, and (iv) covers relationships between the activities. Tab. 1 gives an excerpt of the review findings and how they are supported by CoCoMEP.

Diverse variants of the three parts of CoCoMEP are possible. However, CoCoMEP is appropriate to conduct empirical studies on software evolution as it covers the requirements (see Sec. 2). **R1:** It provides standardized study subject, evolution scenarios, and life-cycle activities. **R2:** This standardization in conjunction with the community offers a structure for collaboration and study replication (see Sec 5). **R3:** CoCoMEP comprises activities and artifacts that correspond to all phases in the system’s life-cycle (life-cycle req.). **R4:** It covers iterations and increments in the development process (evolution req.). **R5:** It provides a concrete setting to qualify the application domain (i.e. supermarket), problem domain (i.e. web-based system) and solution domain (e.g., architecture, code, etc.) of the case (domain req.). **R6:** It supports evaluating the tools necessary to replicate the case, such as implementation/design languages, operating system, or development environments (tool req.).

Finding	Dimension in Fig. 1	CoCoME Platform
Time horizon	design- and run-time	both [18]
Activities	requirements, design, maintenance	use cases [19]/static [20]/dynamic analysis [18]
Artifacts	documentation, UML models, code	design decisions [19], simulation/instrumentation data [18, 21], java code
Relation	req. and impl.	all phases related

Table 1. Supported Review Findings (excerpt)

5 Lessons Learned

In this section, we discuss experiences (wrt. outcomes of Sec. 2 and 3) from applying CoCoMEP in the DFG Priority Programme 1593 which comprises 13 research projects with a focus on long-living systems [6]. The application is exemplified in [18, 19, 20, 21]. CoCoMEP proved to be a suitable knowledge base and supported us in: (i) *Gathering project-spanning understanding on activities and artifacts wrt. evolution.* Mapping the diverse activities and artifacts specific to the single projects within the priority programme into the given life-cycle structure enabled a common understanding of them. Furthermore, common understanding has been supported by a joint communication and documentation infrastructure, i.e. mailing lists, media wiki, SVN repository. The wiki contains all the information about life-cycle activities and related artifacts to be shared and refined among the projects. We use the SVN repository to share source code as well as configuration and documentation artifacts. Based on the life-cycle and infrastructure it was easy to identify and solve uncertainties and misunderstandings among the projects and to create a project-spanning understanding. This is one foundation for research collaboration (i.e. comparability and replication). (ii) *Identifying common artifacts.* Mapping activities and artifacts into the life-cycle allows for identifying artifacts used by diverse projects and relations between artifacts. This is another foundation for research collaborations. (iii) *Reuse of activities and artifacts.* Mapping activities and artifacts into the life-cycle allows for reusing them among the projects and for others. In the priority programme context, the output of activities associated to one project is often reused as an input for activities associated to another project. Using the artifacts in subsequent activities by another project contributes to the evaluation of the artifacts and thus the applied approaches. Furthermore, activities are reused as they are performed by two or more projects. This also contributes to the evaluation of the approaches applied by one projects by comparison to another project. (iv) *Clarifying interfaces between projects.* Project-spanning understanding and knowledge about dependencies between activities and artifacts supports clarifying the interfaces between the single projects. This leads to distribution of responsibilities and thus results in more efficient collaborations. For example, if a required artifact has

already been created by one project, it can often be reused by another project without additional effort. (v) *Using feedback loop*. Including design-time and run-time in the life-cycle allows for analyzing the effects of design decisions at run-time within the same study. This is in contrast to existing studies, which are mostly limited to design and implementation. (vi) *Establishing a technical basis*. CoCoMEP contributed to the development of a common technical basis between the single projects. It supported us in developing tools that interact with each other based on clearly defined interfaces and in configuring common execution environments. Joint tool development and configuration reduces effort for the single projects. Additionally, the integrated tooling eases collaboration while evaluation.

Applying CoCoMEP in the priority programme context, however, showed some potentials for improvement. *Change history of some artifacts is rather short*. Since the priority programme started in 2012, artifacts still face few evolutionary changes compared to ordinary repository mining studies for instance. This is caused by the fact that CoCoME is a research prototype and we do not have the amount of resources (human and financial) involved in real-life development. Nevertheless, as shown by studies in the priority programme, CoCoME provides a sufficient knowledge basis so far for conducting various analysis, e.g. on use cases, decisions, or monitoring and simulation data. We are confident to produce a larger change history in the future as the priority programme continues for further three years and simultaneously CoCoME is applied in a growing number of studies beyond the programme.

6 Conclusion

Based on requirements for collaboration support from related work and a literature review on empirical studies on software evolution, we developed CoCoMEP. The platform consists of three interconnected parts – an established study subject, related evolution scenarios, and a life-cycle covering activities to address the scenarios. Thus, it supports collaboration in and replication of empirical studies by enabling common understanding and reuse of activities and artifacts, interfaces between projects and technical infrastructure, as perceived while applying CoCoMEP in a large research priority programme. In short, CoCoMEP is expected to provide the following benefits to researchers: (i) less effort in scenario definition, study setup and execution, as well as (ii) increased evaluation confidence and (iii) community acceptance by interaction with others. In the future, the subject CoCoME will be further modified to create new and evolve existing artifacts by new evolution scenarios such as the introduction of mobile clients. These scenarios may include parallel evolution and co-evolution of artifacts which are difficult to achieve in most empirical settings.

References

- [1] J. O'Brien and G. Marakas, *Introduction to Information Systems*, 15th ed. McGraw-Hill, 2010.
- [2] M. M. Lehman and L. A. Belady, Eds., *Program Evolution: Processes of Software Change*. Academic Press, 1985.
- [3] D. I. Sjöberg et al., "The future of empirical methods in software engineering research," in *Future of Software Engineering*. IEEE, 2007, pp. 358–378.
- [4] N. Juristo and O. Gómez, "Replication of software engineering experiments," *Empirical software engineering and verification*, pp. 60–88, 2012.
- [5] S. Herold et al., "CoCoME – the common component modeling example," in *The Common Component Modeling Example*. Springer, 2008, pp. 16–53.
- [6] U. Goltz et al., "Design for future: managed software evolution," *CSRD*, pp. 1–11, 2014.
- [7] T. Espinha et al., "Maintenance research in SOA - towards a standard case study," in *CSMR*. IEEE, 2012, pp. 391–396.
- [8] S. Proksch et al., "Towards standardized evaluation of developer-assistance tools," in *RSSE'14*. ACM, pp. 14–18.
- [9] C. Lott and H. Rombach, "Repeatable software engineering experiments for comparing defect-detection techniques," *Empirical Software Engineering*, vol. 1, no. 3, 1997.
- [10] S. Lessmann and B. Baesens, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE TSE*, vol. 34, no. 4, pp. 485–496, 2008.
- [11] F. Shull et al., "Replicating software engineering experiments: addressing the tacit knowledge problem," *Intl. Symposium on Empirical Software Engineering*, pp. 7–16, 2002.
- [12] M. G. Mendonça et al., "A Framework for Software Engineering Experimental Replications," *ICECCS*, pp. 203–212, 2008.
- [13] S. Demeyer et al., "Towards a Software Evolution Benchmark," in *IWPSE*. ACM, 2001, pp. 174–177.
- [14] P. Runeson et al., *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, 2012.
- [15] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University, Tech. Rep., 2007.
- [16] B. P. Lientz and B. E. Swanson, *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley, 1980.
- [17] R. Heinrich et al., "Integrating run-time observations and design component models for cloud system analysis," in *MRT*. CEUR Vol-1270, 2014, pp. 41–46.
- [18] W. Hasselbring et al., "iObserve: integrated observation and modeling techniques to support adaptation and evolution," CAU Kiel, Tech. Rep. 1309, 2013.
- [19] S. Gaertner et al., "Capturing and Documentation of Decisions in Security Requirements Engineering through Heuristics," *SWT-Trends*, vol. 34, no. 1, pp. 21–22, 2013.
- [20] R. Heinrich et al., "Architecture-based analysis of changes in information system evolution," *WSRE, SWT-Trends*, vol. 34, no. 3, 2015.
- [21] R. Heinrich et al., "Run-time architecture models for dynamic adaptation and evolution of cloud applications," CAU Kiel, Tech. Rep. 1503, 2015.

A JVM-based Testing Harness for Improving Component Testability

Weifeng Xu

Department of Computer Science
Bowie State University, Bowie, USA
frank.w.xu@gmail.com

Omar El Ariss

Department of Computer Science & Mathematical Science,
Penn State Harrisburg, PA, USA
oue1@psu.edu

Abstract— Software testing is a critical activity in increasing our confidence of a system under test and improving its quality. The key idea for testing a software application is to minimize the number of faults found in the system. The higher the testability of software, the better our chances to reveal these faults. We introduce a new type of testing harness called GannonJVM that improves the testability of software components. GannonJVM enhances the Java Virtual Machine (JVM) with a predicate analyzer and a bytecode interpreter. Our automated test framework is able to extract and visualize paths from the control flow graph of a given component. We also observe and analyze the predicates in a given path during runtime.

Keywords— Software testing, test harness, bytecode, testing tool, visualization.

I. INTRODUCTION

Testability is a measure of how complex it is to test a software application. The lower the testability of a software, the lower the quality of the generated test cases. On the other hand, the higher the testability of software the better the test cases are. Two major factors that contribute to the testability of software are controllability and observability. Controllability is the ability to access, and the ease of testing the features and functionalities of an application. Observability is the extent to which the output or the observable states of the system assist in the verification of the test results.

A test harness typically refers to a testing framework. It is used to execute test cases and to check whether the actual results match the expected ones. Junit is a popular test harness. For example, the assertion `assertEquals("Isosceles", new Triangle(7,7,6).getTriType())` in JUnit checks whether a triangle object reports the correct triangle type, which is an isosceles for the given three side values: 7, 7 and 6. Although JUnit and other similar unit testing frameworks are capable of executing test cases and checking their results automatically, they usually do not focus on improving the testability (i.e., testing observability and controllability) of software components. For example, JUnit does not provide any feedback that helps the testers to redesign test inputs. It also does not provide useful runtime states for debugging when the test case fails.

To deal with these limitations, we introduce a novel approach to develop a test harness directly on top of the Java Virtual Machine (JVM). We call this approach GannonJVM. Its aim is to improve the testability of Unit Under Test (UUT). Fig. 1 shows the three main components of GannonJVM: (1) a predicate analyzer that is designed to improve the testing observability of UUT. The analyzer defines how the internal states of UUT are monitored and inferred by the knowledge of its external input. It checks the predicate values and monitors their execution paths in terms of the test inputs. (2) a Bytecode interpreter that improves the testing controllability of UUT. The interpreter defines how to stabilize an execution path based on its observation. It is used to adjust the original test input (i.e., seed value) to create additional input that forces the UUT to execute a designated path. (3) Control Flow Graph (CFG) visualizer. It is responsible for automatically determining the layout of a CFG.

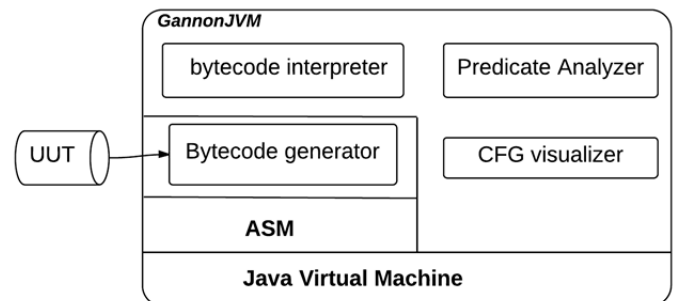


Fig. 1. Overview of the test harness

The rest of this paper is organized as follows: Section II introduces the basic concepts through a running example. Section III and IV describe the predicate analyzer and inference engine. Section V describes the CFG visualizer. Section VI reviews the related work. Section VII concludes the paper.

II. A RUNNING EXAMPLE

We use the classical *Triangle Problem* [1] as a running example to illustrate the test harness with embedded features for observability and controllability to generate test input without fitness functions. Given three positive integers that represent the

lengths of the three sides of a triangle, the *Triangle* program reports the triangle type: *Equilateral* (type 1), *Isosceles* (type 2), *Scalene* (type 3), or *NotATriangle* (type 4). The source code for this problem is shown below:

```
int getTriType (a,b c) {
    if ((a<b+c) && (b<a+c) && (c<a+b)){
        if (a==b && b==c)    return 1;
        else if (a!=b && a!=c &&b!=c) return 3;
        else return 2;}
    else return 4;}

```

Java bytecode is a stack-oriented language, which pops data (operands) from the top of the stack and pushes data back on the top of the stack. The stack is commonly referred to as an operand stack [2]. For example, the bytecode instruction *iadd* pops two integer values from the stack and pushes their sum back to the stack. Two integer values are pre-loaded from a *local variable table* using two *iload* instructions. To facilitate the discussion, the bytecode of *Triangle* instructions are divided (see dashed lines) into 20 blocks shown below [3].

1. <i>iload</i> 1	16. <i>iload</i> 1	28. <i>iload</i> 3
2. <i>iload</i> 2	-----[7]	29. <i>if_icmpeq</i> 35
3. <i>iload</i> 3	17. <i>iload</i> 2	-----[15]
4. <i>iadd</i>	18. <i>if_icmpne</i> 24	30. <i>iload</i> 2
-----[1]	-----[8]	-----[16]
5. <i>if_icmpge</i> 37	19. <i>iload</i> 2	31. <i>iload</i> 3
-----[2]	20. <i>iload</i> 3	32. <i>if_icmpeq</i> 35
6. <i>iload</i> 2	-----[9]	-----[17]
-----[3]	21. <i>if_icmpne</i> 24	33. <i>iconst_2</i>
7. <i>iload</i> 1	-----[10]	-----[18]
8. <i>iload</i> 3	22. <i>iconst_1</i>	34. <i>ireturn</i>
9. <i>iadd</i>	-----[11]	35. <i>iconst_3</i>
10. <i>if_icmpge</i> 37	23. <i>ireturn</i>	-----[19]
-----[4]	24. <i>iload</i> 1	36. <i>ireturn</i>
11. <i>iload</i> 3	-----[12]	37. <i>iconst_4</i>
12. <i>iload</i> 1	25. <i>iload</i> 2	38. <i>ireturn</i>
13. <i>iload</i> 2	26. <i>if_icmpeq</i> 35	-----[20]
14. <i>iadd</i>	-----[13]	
15. <i>if_icmpge</i> 37	27. <i>iload</i> 1	
-----[5]	-----[14]	
-----[6]		

Bytecode instructions have unique properties. First, they have an implicit effect on the stack as each instruction has no explicit named operands. For example, *iadd* (instruction 4) in block 1 does not specify the two operands that will be fetched for integer addition. These values are determined by *iload* 2 and *iload* 3 (instructions 2 and 3) as they are the top two values on the current operand stack. Note that the operand of *iload* points to the index of the local variable table [2]. The local variable table contains bytecode instructions and input parameters after initializing the method invocation. For example when the method is invoked, the three input variables *a*, *b*, and *c* of the triangle program are stored in the first three spots of the local variable table. During execution, *iload* 1, *iload* 2, and *iload* 3 push the values stored in the indices 1, 2, and 3 to the operand stack. Second, predicates with multi-conditions in Java source code are represented by multi-level conditions in bytecode. For example, the multi-condition $(a < b + c) \ \&\& \ (b < a + c) \ \&\& \ (c < a + b)$

in the *Triangle* source code is decomposed into three block sets, i.e., blocks 1 and 2, blocks 3 and 4, as well as blocks 5 and 6. Blocks 2, 4, and 6 are three identical *if_icmpge* statements. They are depicted in Fig. 2 by the CFG of *Triangle* bytecode. This property facilitates observability and controllability by decomposing component conditions into several simple conditions.

A path that is generated from a CFG consists of a sequence of blocks. For example, $p1 = [1] \rightarrow [2] \rightarrow [3] \rightarrow [4] \rightarrow [5] \rightarrow [6] \rightarrow [7] \rightarrow [8] \rightarrow [9] \rightarrow [10] \rightarrow [11]$ is a path for testing if a triangle is *equilateral*. To execute all the blocks in $p1$, the values that participate in the evaluation of predicates [2] [4] [6] [8] and [10] need to be adjusted so that the predicates produce the desired outcomes to reach the last block. Thus, the desired outcome for each predicate in $p1$ should be achieved as $p1$: $[1] \xrightarrow{F} [2] \xrightarrow{F} [3] \xrightarrow{F} [4] \xrightarrow{F} [5] \xrightarrow{F} [6] \xrightarrow{F} [7] \xrightarrow{F} [8] \xrightarrow{F} [9] \xrightarrow{F} [10] \xrightarrow{F} [11]$, where *F* (False) is the expected outcome of the corresponding predicate. Such a path is called a **tagged path**. A tagged path is a sequence of edges that have at least one **tagged edge**, where a tagged edge is defined as $v \xrightarrow{o} u$. The variable *v* is the source block that represents a predicate in a statement, *o* is a **tagged value** for *v*, which represents the desired outcome of *v* (i.e., *true* or *false*), and *u* is the reachable block if the assertion `asserEquals(o, runtime(v))` returns true.

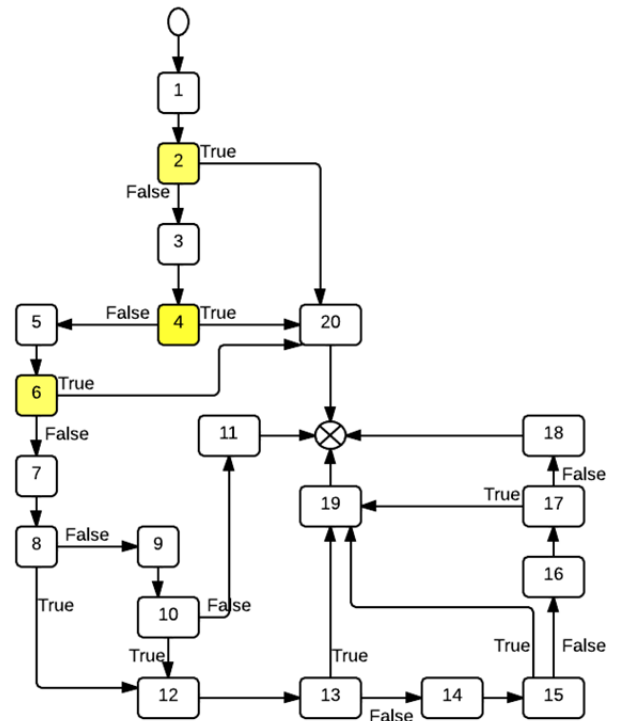


Fig. 2. CFG of the Triangle Problem in bytecode

Table 1 shows some representative tagged paths of the Triangle program based on decision coverage. In tagged path

p12, the tagged edge $[2] \xrightarrow{T} [20]$ indicates that block 2 is a predicate `if_icmpge` and its outcome must be true in order to reach block 20.

Table 1. Tagged paths for Triangle problem

Goal	ID	Path
Equilateral	1	$[1] \xrightarrow{F} [2] \xrightarrow{F} [3] \xrightarrow{F} [4] \xrightarrow{F} [5] \xrightarrow{F} [6] \xrightarrow{F} [7] \xrightarrow{F} [8] \xrightarrow{F} [9] \xrightarrow{F} [10] \xrightarrow{F} [11]$
Isosceles	2	$[1] \xrightarrow{F} [2] \xrightarrow{F} [3] \xrightarrow{F} [4] \xrightarrow{F} [5] \xrightarrow{F} [6] \xrightarrow{F} [7] \xrightarrow{F} [8] \xrightarrow{F} [9] \xrightarrow{T} [10] \xrightarrow{T} [11] \xrightarrow{T} [12] \xrightarrow{T} [13] \xrightarrow{T} [19]$
...
Scalene	8	$[1] \xrightarrow{F} [2] \xrightarrow{F} [3] \xrightarrow{F} [4] \xrightarrow{F} [5] \xrightarrow{F} [6] \xrightarrow{F} [7] \xrightarrow{T} [8] \xrightarrow{T} [12] \xrightarrow{F} [13] \xrightarrow{F} [14] \xrightarrow{F} [15] \xrightarrow{F} [16] \xrightarrow{F} [17] \xrightarrow{F} [18]$
...
NotATriangle	12	$[1] \xrightarrow{T} [2] \xrightarrow{T} [20]$

III. PREDICATE ANALYZER

The predicate analyzer is designed to improve the observability of UUT. It examines bytecode instructions to discover relationships between input variables and variables used in predicates. Discovering relationships relies on variable binding and variable dependency analysis.

The process of binding explicit variables to bytecode instructions is called variable binding. As bytecode instructions have an implicit effect on the evaluation stack, an effective approach is to use instruction tree unit (ITU) as an intermediate representation of instructions. Each ITU is a binary tree, which consists of three nodes, one parent node and two child nodes, as well as an operator (i.e., the opcode of the instruction) between the two children. The child nodes are the explicit named operands. The root is a named intermediate result of the operation. An ITU can be simply represented by a four-tuple (*opcode*, *root*, *leftNode*, *rightNode*). One of the essential characteristics is that ITUs are restricted to the least number of operands (2 in most cases, such as for arithmetic and logic), and these operands must either be constants or locals. For example, for a given Java expression statement $x=a+b+c$, the corresponding two arithmetic ITUs are shown in Fig. 3. Local variables `i0`, `i1`, `i2` are stored in the local variable table and correspond to the variables `a`, `b`, `c` and `x` in the given Java statement. Variables with a “\$” sign are intermediate local variables, e.g., `$i4` is an intermediate variable for holding the value of `i0 + i1` and `$i5` holds the result of `$i4 + i2`. `iadd` is the opcode of the instruction `iadd #index`, where `#index` is the index of the local variable table.

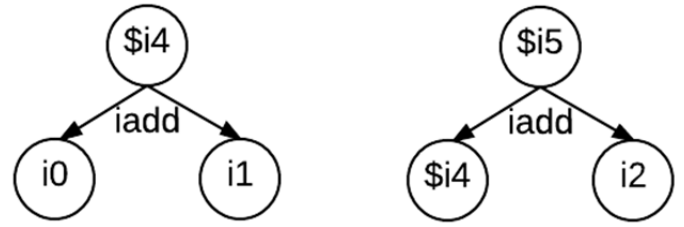


Fig. 3. Two ITUs of Java expression statement $x=a+b+c$

Variable binding is a dynamic process, which builds the ITUs along with the execution of bytecode. We utilize an additional stack, called *variable binding stack*, and a variable table, called *variable binding table*, to bind variables to instructions. We gave them these names to distinguish them from the operand stack and the local variable table specified by the JVM specification. The *variable binding stack* and *variable binding table* work very similar to the JVM operand stack and the local variable table except that 1) the *variable binding stack* and *variable binding table* store the names of the bytecode intermediate variables instead of the operands for tracking intermediate variables, and 2) each element of the *variable binding table* also has a reference point to the root of the ITU containing itself.

Variable binding during instruction execution in the JVM works as follows: 1) whenever an instruction pushes a value into the operand stack, and the value is loaded from the local variable table, the index of the value in the local variable table is used as the intermediate local variable name. This index is pushed into the *variable binding stack*. Otherwise, a new generated unique ID is used as the name and is pushed into the *variable binding stack*. 2) whenever an instruction pops a value from the operand stack, the top of the *variable binding stack* is removed as well. The popped intermediate local variable names are used for constructing the ITUs. Note that for the purpose of dependency analysis, we build ITUs only for instructions that produce an effect on the operand stack and are influenced by the effect, i.e., instructions that produce and use intermediate variables. Therefore, ITUs are categorized into two groups: expression ITU and predicate ITU. Expression ITUs are built from expression instructions [4] producing intermediate variables, including load, arithmetic, and logic instructions. Predicate ITUs are built from predicate instructions using the intermediate variable to compute the tagged values, including all `if_*` instructions. The algorithm can be applied for binding other instructions. Fig. 4 shows the variable binding results (i.e., the two ITUs) for the tagged path p12: $[1] \xrightarrow{T} [2] \xrightarrow{T} [20]$ in Table 1. The block list is a variable binding table. The first three variables, `i0`, `i1`, and `i2`, are the names of the input parameters. `$i10` and `$i11` are intermediate variables pointing to the root of the two ITUs shown in Fig. 4. The letter “i” is added before the generated ID as part of variable name for readability.

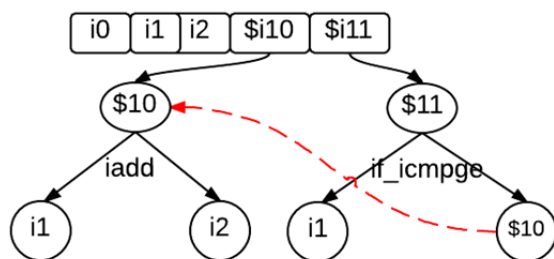


Fig. 4. Variable binding for path P12

Variable dependency analysis is the process of backtracking input variables for a given intermediate bytecode variables for making the assertion `asserEquals(o, runtime(v))` to be true. Again, considering the simple tagged path p12: the goal is to find a test input to execute this path (i.e., find a triangle type of “*NotATriangle*”). As $[2] \xrightarrow{T} [20]$ is the only tagged edge, the path will be covered if a test input forces the constraint in the instruction `if_icmpge` to be *true* (statement 5 is the only instruction in block [2]). The predicate ITU (`if_icmpge, $i11, $i1, $i10`) indicates that to generate a test input to cover p12, however, we need to determine the input variables that are associated with `$i10`. The association will allow the proposed system to backtrack the input variables so that they can be adjusted to meet the constraint. It is not difficult to see that `$i10` (shown in the expression ITU `iadd` on the left of Fig. 4) is associated with input variables `i1` and `i2` by backtracking `$i10` in the predicate ITU on the right. Variable dependency can be graphically captured using a Variable Dependency Tree (VDT). A VDT consists of a set of ITUs, where the root and each intermediate node are intermediate variables, and all leaves are the bytecode input variables. The algorithm below describes the procedure for building VDTs from ITUs. The algorithm recursively expands child nodes containing intermediate variables with ITUs. The algorithm below describes the procedure for building VDTs from ITUs. The algorithm recursively expands child nodes containing intermediate variables with ITUs. The red dashed line shown in Fig. 4 indicates a backtracking relation of `$i10`.

Algorithm: Building VDTs

Inputs: VBT: A variable binding table

Outputs: VDT: A variable dependency tree

procedure buildVDTs(VBT)

```

for each element E of VBT
  (opeCode, root, leftNode, rightNode) ←
  E.getITU()
  if leftNode/rightNode of the ITU containing
  intermediate variable
    newITU ← find a new ITU based on leftNode or
    rightNode
    Point from leftNode/rightNode to newITU
  end if
end for
end procedure

```

IV. BYTECODE INTERPRETER

Bytecode interpreter aims to improve testing controllability of UUT, i.e., how to control the predicate evaluation results to force a given path to be executed at run-time. Note that the evaluation results are determined by the input, where the rule-based inference engine provides input changing guidelines.

Bytecode interpreter controls the order of which bytecode instruction will be fetched and executed. It reads each bytecode instruction and returns the evaluation result. It mainly consists of a program counter, which points to the next instruction to be fetched and executed, a local variable table, and an operand stack. In addition, a Java stack is needed for method invocations. Each element of the Java stack is a Java frame, which stores execution status. To make the interpreter more flexible, we utilize a factory design pattern to encapsulate instruction creation and a strategy pattern to encapsulate the execution algorithm in each instruction. A snapshot of the implementation of `BIFicmpge` instruction is shown below. The execution method implements the abstract method defined in the `Instruction` class. This comparison instruction pops two values from the operand stack and returns the predicate result. It is worth noting that the bytecode input parameters are stored at the beginning of the local variable table. They will be fetched for UUT interpretation. It is not difficult to overwrite them with new generated input in order to make the input generating process automatic. Along with the predicate analyzer and rule inference engine, this overwriting mechanism makes the UUT running until a given path is executed.

```

public class IFicmpge extends Instruction {
  @Override
  public Object execute(JavaFrame frame) {
    Stack<Integer> opStack =
    frame.getOperandStack();
    Integer rightValue = (Integer) opStack.pop();
    Integer leftValue = (Integer) opStack.pop();
    boolean result=rightValue>= leftValue;
    return result;
  }
}

```

The Bytecode interpreter then collaborates with the *Bytecode generator*. Compiled Java class files are in the form of hexadecimal. Therefore, ASM [5] is utilized to convert hexadecimal numbers to readable bytecode instructions. ASM is a very small and very fast Java bytecode manipulation framework supported by Open Solutions Alliance.

V. CONTROL FLOW GRAPH VISUALIZER

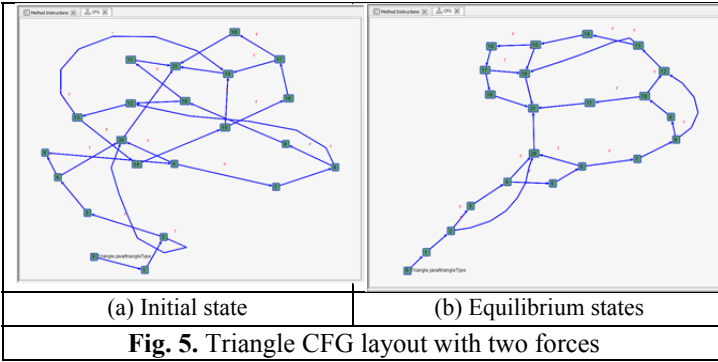
A directed graph $G = \{V, E\}$ consists of two types of elements V and E , where V is a set of vertices and E is a set of edges. A Control Flow Graph (CFG) is a graph with some special vertices and edges: 1) it has source and sink vertices and 2) it consists of loops and jumps. Control Flow Graph (CFG) visualizer is responsible for determining the layout of a CFG

automatically. CFG visualizer needs to solve three challenges: 1) how to calculate the layout of graph if we treat CFG is a general type of graph, 2) how to handle with two special vertices, i.e., source and sink, and 3) how to determine two special edges of CFG, i.e., loops and jumps.

A. Visualizing CFG as A Normal Graph

Force-directed algorithms are the most flexible and popular algorithms for calculating layouts of simple undirected graphs. These algorithms calculate the layout of a graph using only information contained within the structure of the graph itself. For a given directed graph $G = \{V, E\}$, a force-directed algorithm models edges as springs and vertices as charged particles. Springs represent attractive forces based on Hooke's law, which are used to attract pairs of connected vertices towards each other. Charged particles represent repulsive forces based on Coulomb's law, which are used to separate all pair of vertices. Force is represented as a vector, which includes a magnitude and direction. In a force-directed algorithm, we start with assigning a random position for each vertex. Then each vertex applies the attractive and repulsive forces. This will cause the vertex to move to a new position. The calculating and moving activities repeat until the graph reaches equilibrium states. In equilibrium states for a given graph, edges tend to have uniform length because of the spring forces, and nodes that are not connected by an edge tend to be drawn further apart because of the electrical repulsion.

Fig. 5 shows the automated layout calculation using the attractive and repulsive forces on the Triangle problem CFG. Vertices in Fig. 5 (a) are assigned random positions. Fig. 5 (b) shows the equilibrium states of the CFG.



The force-directed algorithm is defined below and is based on Eades' idea [6]:

```

Algorithm SPRING(G: graph)
  Place vertices of G in random locations
  Repeat M times
    Calculate the force  $\vec{F}(v)$  on each vertex
    Move the vertex based on force on vertex
    Draw graph on screen
  End of Algorithm
  
```

The force $\vec{F}(v)$ is defined as:

$$\vec{F}(v) = \sum_{(u,v) \in V \times V} \vec{H}_{uv} + \sum_{(u,v) \in E} \vec{C}_{uv} \quad (1)$$

Where \vec{H}_{uv} represents the attractive force between two connected vertices, u and v , calculated based on Hooke's law. \vec{C}_{uv} represents the repulsive force between vertices u and v , and is calculated based on Coulomb's law.

B. Positioning Source and Sink Vertices

The control flow graph $G(f) = \{V, E, v_{in}, v_{out}\}$ of a function f has two additional vertices, source and sink vertices, referred as v_{in} and v_{out} , respectively. A source vertex is a vertex with indegree zero, while a sink vertex is a vertex with outdegree zero. The control flow graph of an empty function, i.e., a function without any statements consists of $V = \{v_{in}, v_{out}\}$ and $E = \{(v_{in}, v_{out})\}$.

Unlike the layout solution shown in Fig. 5, traditionally, all vertices of a CFG are arranged in the form of top-to-bottom where v_{in} and v_{out} are placed on the top and bottom positions, respectively. In order to rearrange v_{in} and v_{out} in Fig. 5, the third force, named *Earth Gravitational Force*, is added to formula (1). The gravity of Earth, denoted as \vec{T} , refers to the acceleration that the Earth imparts to objects on or near its surface.

The Earth Gravitational Force is defined as:

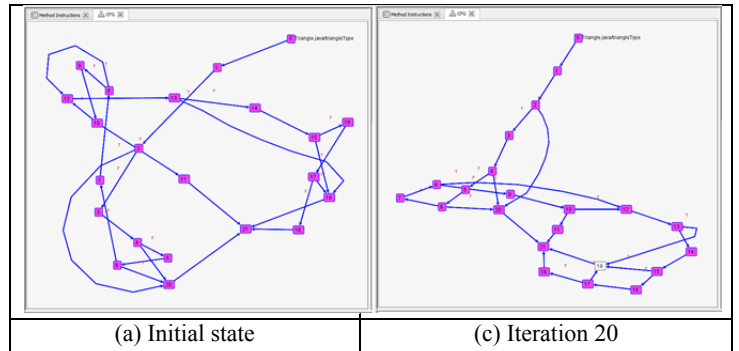
$$\vec{T}(v) = mg \quad (1)$$

Where, m is the mass of the vertex and g is the gravitational content.

The new formula for handling source and sink vertices is now defined as:

$$\vec{F}(v) = \sum_{(u,v) \in V \times V} \vec{H}_{uv} + \sum_{(u,v) \in E} \vec{C}_{uv} + \sum_{(u) \in E} \vec{T}_u \quad (2)$$

Fig. 6 shows the automated calculated *Triangle* CFG layout with the additional Earth gravitational force. Fig. 6 (a) (b) (c) (d) illustrates the evaluations of the CFG layout.



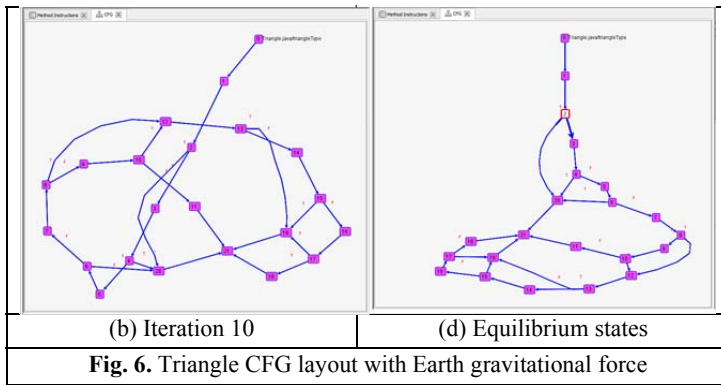


Fig. 6. Triangle CFG layout with Earth gravitational force

C. Positioning Loops and Jumps Edges

There are two types of special edges, loops and jumps (i.e., loop and if-else statements) in a CFG. For example, v_2 in Fig. 6 is a predicate node containing an if-else statement. Without an appropriate positioning algorithm, the edge (v_2, v_{20}) will be a straight line. Positioning such special edges need (1) Identifying dominator relationships: In a CFG graph, a vertex v dominates another node w if and only if every directed path from v_{in} to w in the CFG contains v . The dominators of node w is defined as $\text{dom}(w) = \{v \mid v \text{ dominates } w\}$. For example, $\text{dom}(v_{20}) = \{v_0, v_1, v_2\}$. (2) Identifying special edges: The node $v_{\text{shortest}} = v \in \text{dom}(w)$ has the shortest path from v to w , where v is the start node and w is the end node, i.e., the special edge is defined as (v_{shortest}, w) , and (3) Adding invisible vertices to special edges: The number of invisible vertices equals to the number of vertices from v_{shortest} to w .

VI. RELATED WORK

Various testing harnesses have been explored to monitor the runtime state of UUT. These tools mainly fall into two categories: aspect-oriented approaches and symbolic execution based approaches. MOP [7] is a Monitoring-Oriented Programming (MOP) framework, which automatically generates monitors from the specified properties and then integrates them together with the user-defined code into the original system. In the implementation, parametric specifications are translated into AspectJ [8] code, and then weaved into the application using off-the-shelf AspectJ compilers. Tracematches [9] is another aspect-oriented trace-matching tool to observe the execution of a base program; when certain actions occur, the aspect runs some extra code of its own. Java Pathfinder (JPF) [10][11] is a system to verify executable Java bytecode programs. It is based on symbolic execution for test case generation. The core of JPF is a Java Virtual Machine that is also implemented in Java. JPF executes normal Java bytecode programs and can store, match and restore program states. KLOVER [12] is similar to JPF. It executes and monitors the states of running C++ program in the form of LLVM bytecode. GannonJVM implements the features of testing observability and controllability, which monitors, interpreters and controls the Java bytecode instructions directly using a stack-based approach.

VII. CONCLUSION

This paper presents a novel approach to embed two testability features, including testing observation and testing control features. We also introduce a CFG visualization in Java Virtual Machine (JVM) for building a new testing harness to facilitate software testing. The implementation of GannonJVM, a demo video, and the *triangle* example are publicly available¹.

VIII. ACKNOWLEDGMENT

We thank Syed Aqeel Raza and Bader Aldawsari for assistance with the implementation of the system.

REFERENCES

- [1] P. C. Jorgensen, *Software Testing: A Craftman's Approach*, 3rd ed., Auerbach Publications, 2008.
- [2] T. Lindholm, F. Yellin, G. Bracha and A. Buckley, *Java Virtual Machine Specification, Java SE 7 Edition*, Boston, USA: Addison-Wesley Professional, 2013.
- [3] J. Zhao, "Analyzing Control Flow in Java Bytecode," in *16th Conference of Japan Society for Software Science and Technology*, Japan, 1999.
- [4] R. Vallee-Rai and L. J. Hendren, "Jimple: Simplifying Java Bytecode for Analyses and Transformations," Sable Research Group, School of Computer Science, McGill University, Montreal, Canada, 1998.
- [5] O. Consortium, "ASM," [Online]. Available: <http://asm.ow2.org/>. [Accessed 23 08 2013].
- [6] P. Eades, "A heuristic for Graph Drawing," *Congressus Numerantium*, vol. 160, no. 42, p. 149, 1984.
- [7] F. Chen and G. Rosu, "MOP: An Efficient and Generic Runtime Verification Framework," in *Object-Oriented Programming, Systems, Languages & Applications*, Nashville, Tennessee, 2007.
- [8] "AspectJ," Eclipse Foundation, [Online]. Available: <http://eclipse.org/aspectj/>. [Accessed 10 Sep 2013].
- [9] C. Allan, P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, O. Lhotak, O. d. Moor, D. Sereni, G. Sittampalam and J. Tibble, "Adding Trace Matching with Free Variables to AspectJ," in *Object-Oriented Programming, Systems, Languages & Applications*, San Diego, California, 2005.
- [10] N. A. R. Center, "Java Pathfinder," NASA Ames Research Center, [Online]. Available: <http://babelfish.arc.nasa.gov/trac/jpf>. [Accessed 27 1 2014].
- [11] C. S. Pasareanu, et al., "Symbolic PathFinder: Integrating Symbolic Execution with Model Checking for Java Bytecode Analysis," *Automated Software Engineering*, vol. 20, no. 3, p. 391-425, 2013.
- [12] G. Li, I. Ghosh and S. P. Rajan, "KLOVER: A Symbolic Execution and Automatic Test Generation Tool for C++ Programs," in *23rd International Conference on Computer Aided Verification*, Snowbird, Utah, 2011.

¹ Implementation and example: <http://perceval.gannon.edu/xu001/research/GannonJVM/>. Source code: [git@github.com:Gannon-University/GannonJVM.git](https://github.com/Gannon-University/GannonJVM.git). Demo video: <https://www.youtube.com/watch?v=Ey4JfVhhHQg>.

Detecting Reporting Anomalies in Streaming Sensing Systems

Shiree Hughes¹, Yuheng Du², and Jason O. Hallstrom¹

¹I-SENSE, Florida Atlantic University, {shughes2015, jhallstrom}@fau.edu

²School of Computing, Clemson University, yuhengd@clemson.edu

Abstract

Sensor networks must be monitored to identify and correct problems as they occur. We present a comparison of two approaches to monitoring deployed sensors. The first relies on configuration parameters to define expected reporting behavior. The second automatically identifies normal reporting patterns based on a combination of configuration parameters and an analysis of reporting times. Using these patterns, the system notifies personnel of possible malfunctions. We present empirical evaluations in the context of the Intelligent River[®] system [11].

1. Introduction

Wireless sensor networks are often used to collect data over large geographic areas. Environmental factors, ranging from changes in cellular or satellite signal strength to equipment damage can disrupt data collection. It is important to identify when sensors are malfunctioning to avoid prolonged data loss. Equally important is the ability to ignore routine or minor fluctuations in reporting behaviors to avoid spurious maintenance notifications.

We present two approaches to monitoring the behavior of deployed sensors. The *static* approach is based on user configuration data that captures the expected reporting period of each sensor. The *adaptive* approach relies on an on-line time-series analysis of reporting times. The ability to automatically identify reporting patterns enables the determination of whether a sensor is more likely to be damaged, versus exhibiting normal variation in reporting behavior.

We evaluate the performance of both approaches using historical data from 122 devices reporting at various rates over a six-month period within the Intelligent River[®] network. Many of these devices are enclosed in bouys designed to keep them afloat and protect them from the water. We explore the effects of various parameters on the sensitivity of the monitoring system.

2. Related Work

Many authors have considered methods to identify sensor malfunction and proposed various solutions [5, 6, 7, 13, 17], most concerned with the validation of received data.

Ramanathan et al. [10] present Sympathy, a tool to detect and debug sensor failures. Similar to our static approach, Sympathy assumes an expected time period in which a sensor should report, and then flags the sensor if it does not report within the expected time. Sympathy requires each sensor to transmit a set of metrics, including information on network connectivity, packet reception rate, and packet corruption rate. When a failure is detected, the metrics are analyzed to determine cause and location. Sympathy does not adapt to periodic network delays and will repeatedly send notifications during periods of abnormal network behavior.

Shebaro et al. [12] focus on determining the cause of packet loss. Their method employs comparisons of RSSI and LQI values. While link quality can be a good indicator of malfunction, information detailing when sensors are expected to report is also needed to diagnose sensor failure.

Duche and Sarwade [2] use round-trip delay (RTD) to detect malfunction in multi-hop scenarios. When a sensor fails to transmit, the RTD of the sensor approaches infinity. By defining an initial threshold for comparison, a malfunctioning sensor can be identified if its RTD is greater than the threshold. Each sensor has several paths it may transmit over, and each path may have a different RTD. Their approach assumes a circular topology.

Guo et al. [4] concentrate on the detection of faulty data. Their approach considers both sensor data and location and assumes that all sensors observe and report data on the same event. Sensors are designated as faulty if they do not adhere to distance monotonicity. The goal is to create a list of sensors reporting faulty data ranked by probability, not to notify personnel of sensors' failure to report.

Warriach et al. [14] propose a hybrid fault detection approach that combines rule-based, linear least-squares estimation and hidden Markov methods to identify data faults. The approach does not detect the absence of data.

Bartzoudis and McDonald-Maier [1] describe a method to validate sensor readings. Their approach compares prior readings to current readings to identify improbable discrepancies, such as drastic temperature changes in a short time period. Reported values are also verified to be within the operating limits specified by the manufacturer. The purpose is to identify data faults, not to identify reporting failures.

Zang et al. [16] use a combination of principal component analysis and wavelet analysis to detect sensor failure

in small and medium-scale networks. Detection of a malfunctioning sensor is dependent upon data collected from neighboring sensors. Our goal is to identify malfunctioning sensors independent of other sensors within a network.

Friend et al. [3] are concerned with commercial credit-card fraud. We take inspiration from methods of fraud detection in this context to identify malfunctioning sensors. Like people, each sensor has an identifiable reporting pattern which can be used to detect changes in behavior. Commercial fraud detection is achieved by comparing current observations with expected values derived from previous observations. Friend et al. find that a customer's transactions can be represented by a standard bell curve. The average value of a customer's transactions defines the peak of the curve. Each standard deviation away from the average contains transactions that are more likely to be fraudulent. The authors find that only 1% of transactions are more than 3 times the standard deviation. We use this principle and apply it to sensor reporting periods in the adaptive approach to determine sensor malfunction.

3. The Static Approach

We now detail our implementation of the static approach.

3.1. Configuration

Configuration data is specified using JSON data stored in MongoDB. Sensors are separated into groups based on deployment area, and each group is represented as a JSON object, such as the example shown in Listing 1. Six properties are defined for each group: `groupID` is a unique group identifier. `members` specifies a list of sensors included in the group, each identified by ID. `expectedInterval` specifies the maximum number of seconds that may pass between reports from sensors within the group. `notificationTime` specifies the number of seconds to wait between personnel notifications while a sensor is malfunctioning. `maxNotifications` specifies the maximum number of notifications that should be sent to network personnel while a sensor is malfunctioning. The configuration data also stores *notification* group objects, which define personnel contact information. A sample notification group object is shown in Listing 2. The `notificationGroupID` in Listing 1 specifies the identifier corresponding to the notification group defined in Listing 2, declaring the list of personnel to be contacted if a malfunction is identified within the sensor group.

3.2. Implementation

The monitoring service was developed as part of the Intelligent River[®] system [15], a network of sensing devices deployed throughout the Savannah River Basin to support water management and agricultural applications [11]. Each sensor transmits environmental data through the network using RabbitMQ [9], an open source implementation

```
1 {"GroupID":"Sensor-Group-1",
2  "members":["sensor-1",
3           "sensor-2",
4           "sensor-3"],
5  "notificationGroupID":"Notification-Group-1",
6  "expectedInterval":900,
7  "notificationTime":1500,
8  "maxNotifications":5 }
```

Listing 1. Sample Sensor Group

```
1 {"notificationGroupID":"Notification-Group-1",
2  "addresses":["user1@domain.com",
3             "user2@domain.com"] }
```

Listing 2. Sample Notification Group

of the AMQP standard [8]. In the Intelligent River[®] system, RabbitMQ receives data reports routed through dedicated queues. The sensor monitoring service uses one of these queues to obtain device readings, and to monitor time-stamps. The implementation is developed in Java and consists of three main classes: `MessageHandler`, `StatusChecker`, and `Notifier`.

3.2.1. MessageHandler

The `MessageHandler` class consumes messages from a dedicated message queue. Each message is parsed to determine the identity of the reporting device and the time-stamp of the report. Instances of the class maintain a local hashmap, mapping from sensor ID to a data object containing the time-stamp of the most recent report, the time-stamp of the most recent notification, the number of notifications sent, and the current status of the sensor (i.e. alive or dead).

3.2.2. StatusChecker

The `StatusChecker` class determines if a sensor's status should be updated. At startup, a `StatusChecker` thread is created for each sensor group identified in the input configuration. Listing 3 summarizes the core logic used in each thread. `StatusChecker` wakes every `expectedInterval` seconds (line 2) and queries `MessageHandler` for the latest time-stamp associated with each sensor within the group (lines 3–4). The system then calculates the most recent reporting interval, `timeDifference` (line 5). If the reporting interval is not within the `expectedInterval`, the sensor's status is set to dead (lines 6–7). After marking a sensor as dead, `StatusChecker` determines the interval between the last time of notification and the current system time (line 8). If the interval is more than `notificationTime` and the number of notifications sent is less than `maxNotifications`, a `Notifier` instance is created to handle notification (lines 9–10).

`StatusChecker` also detects revivals; i.e. sensors marked as dead which begin to report. When

```

1 while(true) {
2     wait(expectedInterval);
3     for each sensorID in members {
4         sensor = hashmap.get(sensorID);
5         timeDifference = currentTime-sensor.lastReportTime;
6         if(timeDifference>expectedInterval) {
7             sensor.status(dead);
8             notifyInterval = currentTime-sensor.notifyTime;
9             if(notifyInterval < notificationTime &&
10                sensor.notificationsSent<maxNotifications) {
11                 new Notifier(sensor);
12                 sensor.notificationsSent++;
13                 sensor.notifyTime = currentTime;    }
14             } else if (sensor.status == dead) {
15                 sensor.reset();
16                 new Notifier(sensor);    }    }    }

```

Listing 3. StatusChecker Algorithm

StatusChecker identifies such a sensor, the sensor's status, notification count, and time of last notification are reset, and a notification is generated (lines 13–15).

3.2.3. Notifier

A Notifier instance is created to notify personnel of node malfunctions and revivals. The sensor's data object is passed to the object at construction. The Notifier object then retrieves the addresses within the notification group identified by notificationGroupID and composes a message containing the sensors' status, identifier, and time of last report. Malfunction notifications also contain the notification count to remind network personnel of how many messages they have already received.

4. The Adaptive Approach

Although each sensor is programmed to transmit at a specific interval, environmental and other factors introduce regular fluctuations in observed reporting behavior. In Intelligent River® deployments, wind, rain, dam discharge events, and other factors can cause bouys to drop below the water surface, preventing the enclosed sensors from transmitting via cellular or satellite. We have observed that in most locations, changes in transmission intervals due to temporary environmental factors follow regular patterns.

Since each sensor deviates from its programmed reporting interval differently, it is difficult to set a uniform maximum time interval that encompasses the behavior of all sensors. It would also be tedious to manually update the expected behavior of every sensor within the network. To avoid excessive notifications without requiring significant effort from network administrators, we developed an approach that detects normal variance and estimates a best-fit interval for each sensor within a given group.

4.1. Configuration

A sample configuration showing the definition of a sensor group in the adaptive approach is shown in Listing 4. We continue to use the parameters from the static approach, with three additions. numberOfStdDevs repre-

sents the number of standard deviations from the average reporting interval considered to be an acceptable deviation. windowSize represents the number of previous reporting intervals to be considered when calculating metrics such as standard deviation and mean. decayConstant represents how quickly a sensor is expected to return to normal reporting behavior after experiencing failure.

```

1 {"GroupID":"Sensor-Group-1",
2  "members":["sensor-1",
3            "sensor-2",
4            "sensor-3"],
5  "notificationGroupID":"Notification-Group-1",
6  "expectedInterval":900,
7  "notificationTime":1500,
8  "maxNotifications":5,
9  "numberOfStdDevs":3,
10 "windowSize":25,
11 "decayConstant":12 }

```

Listing 4. Sample Sensor Group (adaptive)

4.2. Implementation

In the *adaptive* implementation, StatusChecker is modified, and an Analyzer class is added.

4.2.1. Analyzer

Each time a sensor reports, an instance of the Analyzer class calculates the interval between the two most recent reports and stores the interval in a circular buffer. The size of the buffer is determined by windowSize. Each sensor has a dedicated Analyzer object stored in the hashmap discussed in Section 3.2.1. The object maintains the buffer and provides mean and standard deviation methods.

4.2.2. StatusChecker

In the adaptive approach, we define three sensor states: (i) *initial*, indicating that sufficient data to accurately predict the sensor's behavior has not yet been collected; (ii) *normal*, indicating that the sensor is reporting within the expected range; and (iii) *abnormal*, indicating that the sensor is reporting outside its expected range.

As in the static approach, each StatusChecker thread wakes every expectedInterval seconds to check the status of the sensors within its associated group. A new check() method is invoked at wake-up to determine the status of the sensor, as discussed in the next paragraphs. Each time the MessageHandler object receives a report from a sensor, the associated StatusChecker updates its timer, its Analyzer object, and its last timestamp. This is handled in the update() function of MessageHandler. As in the static approach, if a sensor is identified as dead or revived, the system takes the necessary notification steps. Listing 5 shows the update() method for the adaptive version of StatusChecker. timeDifference represents the observed interval between consecutive sensor reports (line 1). average represents the average of the intervals stored in the sensor object's buffer (line 3). stddev represents the standard devi-

```

1 timeDifference = currentReport - sensor.lastReport;
2 sensor.addToBuffer(timeDifference);
3 average = Analyzer.getAverage(sensor);
4 stddev = Analyzer.getStdDev(sensor);
5 allowable = average + stddev*numberOfStdDevs;
6 switch(sensor.state) {
7     case(INITIAL):
8         sensor.resetTimer(configExpectedInterval);
9         break;
10    case(NORMAL):
11        if(timeDifference>allowable) {
12            sensor.state = ABNORMAL;
13            sensor.expectedInterval = timeDifference;
14            sensor.resetTimer(timeDifference);
15        } else { sensor.resetTimer(allowable); }
16        break;
17    case(ABNORMAL):
18        if(timeDifference<sensor.expectedInterval){
19            sensor.expectedInterval -= (
20                (sensor.expectedInterval-allowable) /
21                decayConstant);
22            if(sensor.expectedInterval < allowable) {
23                sensor.state = NORMAL;
24            }
25        } else { sensor.expectedInterval = timeDifference; }
26        sensor.resetTimer(sensor.expectedInterval);
27        break;
28 }
29 sensor.lastReport = currentReport;
30 if(sensor.status == dead) {
31     sensor.status == alive;
32     new Notifier(sensor); }

```

Listing 5. StatusChecker update () Method

ation of these intervals (line 4). `allowable` represents the allowable maximum interval between normal sensor reports (line 5).

Initial State. Initially, the Analyzer’s buffer is empty. Before the buffer contains the required number of intervals, the average and standard deviation may not reflect the longer-term values and may fluctuate significantly. In this state, the system expects to receive the next report within the period specified in the configuration (line 8). Regardless of state, the time-stamp of the most recent report is updated to reflect the new time-stamp (line 27). Finally, `update()` checks if the reporting sensor is flagged as dead and notifies network personnel of revivals (lines 28–30).

Listing 6 shows the logic for `check()` in the *adaptive* version of StatusChecker. On wake-up in the *initial* state, the system compares the time difference between the current system time and the last known reporting time to the `expectedInterval` specified in the configuration file (lines 6–11). If the sensor has not reported within the `expectedInterval`, a notification is sent (lines 7–8). Next, the system checks if Analyzer contains enough data to move the sensor to the normal state (lines 9–10).

Normal State. Consider the `update()` logic for a sensor in the *normal* state, shown in Listing 5 (lines 10–16). The method queries the corresponding Analyzer for the average and standard deviation of the sensor’s reporting intervals (lines 3–4). The maximum reporting interval is calculated based on the current average and the

`numberOfStdDevs` parameter, which controls the sensitivity of acceptability. The system places the sensor in the *abnormal* state if the observed reporting interval is outside of `allowable`, and the timer corresponding to that sensor is then reset (lines 11–14). When the sensor is moved to the *abnormal* state, the sensor’s `expectedInterval` is set to the observed reporting interval (line 13). This outlying interval serves as an estimate of the delay the sensor will experience while in the *abnormal* state.

Once a sensor enters the *normal* state, the `check()` method follows the actions in Listing 6 on wake-up (lines 12–16). `allowable` is calculated as before, in the `update()` method. The system then checks if the time difference between the current system time and the last known reporting time is outside `allowable` (line 13). If so, the sensor object is updated to reflect its change in status, and a notification is sent (lines 14–15).

Abnormal State. When a sensor reports outside of its `expectedInterval`, it is placed in the *abnormal* state. In this state, the system assumes the sensor is malfunctioning and does not expect it to report within the calculated `expectedInterval`. The `update()` logic for the *abnormal* state is shown in Listing 5 (lines 17–25). `allowable` is calculated as in the *normal* state. If the observed reporting interval is less than the `expectedInterval` stored in the sensor object, it is assumed that the sensor is recovering, and the stored value is adjusted to reflect this (lines 18–19). The rate at which the adjusted interval is decreased is dependent on `expectedInterval`, `allowable`, and `decayConstant`. The rate is calculated as the difference between the sensor object’s `expectedInterval` and `allowable`, divided by `decayConstant` (line 19). `decayConstant` enables network administrators to control how quickly abnormal behavior is expected to converge to normal behavior. If a sensor reports outside of its `expectedInterval`, `expectedInterval` is raised to the outlying value (line 23). Once `expectedInterval` is within `allowable`, the sensor is returned to the *normal* state (lines 20–22). This process ensures that a sensor is consistently demonstrating normal behavior before being returned to the *normal* state.

The `check()` method for the *abnormal* state follows the actions shown in Listing 6. The system checks if the difference between the current system time and the last known reporting time is within `expectedInterval`, and notifies network personnel accordingly (lines 17–20).

5. Evaluation

We collected data over a six-month period for 122 unique devices in various locations with various reporting intervals. These devices provided a total of 2,648,267 transmissions to analyze. First, we perform a baseline evaluation of the noti-

```

1 timeDifference = currentTime-sensorID.lastReportTime;
2
3 ... average, stddev, and allowable are calculated as in
  Listing 5 on lines 3, 4, and 5 respectively ...
4
5 switch(sensorID.state) {
6   case(INITIAL):
7     if(timeDifference > configExpectedInterval) {
8       new Notifier(sensor); }
9     if(sensorID.isFull()) {
10      sensorID.state = NORMAL; }
11    break;
12   case(NORMAL):
13     if(timeDifference>allowable) {
14       sensor.status = dead;
15       new Notifier(sensor); }
16    break;
17   case(ABNORMAL):
18     if(timeDifference > sensor.expectedInterval) {
19       new Notifier(sensor); }
20    break; }

```

Listing 6. StatusChecker check() Method

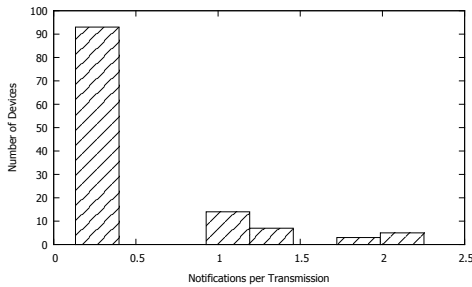


Figure 1. Static Approach Performance

fication tool’s performance using the static approach. Next, the effects of the 3 configuration parameters introduced in the adaptive approach are evaluated. We then compare the performance of the adaptive and static approaches in terms of the number of notifications generated and the ability to customize performance based on user needs.

5.1. Static Approach

For the static method, `expectedInterval` was set for each device by personnel familiar with the reporting behavior of the device. The number of notifications generated was then compared to the number of transmissions sent by each device. The findings are shown in Figure 1. Figure 1 shows the distribution of notification rates across devices. Devices were placed in bins of width .265 (notifications per transmission) based on notification rate. The x-axis represents the number of notifications sent per transmission. The y-axis represents the number of devices with a notification rate within the given range. 93 out of 122 devices (76%) exhibited a notification rate of less than .265 notifications per transmission. This means that for the majority of devices, out of every 100 transmissions, approximately 26 or less of the transmissions were flagged as irregular, generating a notification. The other 29 devices exhibited notification rates greater than .795 notifications per trans-

mission. Devices with a notification rate greater than .795 notifications per transmission generated at least 79 notifications for every 100 transmissions. Due to the configuration of `notificationTime`, which enables multiple notifications to be generated during a single interval between transmissions, some devices had a rate greater than 1 notification per transmission. The maximum rate observed was 2.12 notifications per transmission.

5.2. Adaptive Approach

To evaluate changes in notification rate as a function of changes to the three configuration parameters used in the adaptive monitoring approach, we first determine the value of `windowSize` for each device corresponding to a 24-hour period. Each device is placed in one of seven groups based on its programmed reporting interval: 6-second, 5-minute, 10-minute, 15-minute, 20-minute, 30-minute, and 1-hour. For each group, we run the adaptive approach using various combinations of `numStdDevs` and `decayConstant`. We varied `numStdDevs` between 0 and 5, and `decayConstant` between 0 and $1.4e6$. This range was chosen to explore expected time to recovery ranging from instantaneous (`decayConstant` equal to 0) up to 6 months (`decayConstant` equal to $1.4e6$).

Figure 2 summarizes the effects of these two parameters on notification rate for 3 of the groups. The graphs were subdivided to provide a more detailed understanding of performance. Figures 2(a), 2(b), and 2(c) show the effects of `decayConstant` in the range of 0 to 1000, while Figures 2(d), 2(e), and 2(f) show its effects in the range of 1000 to $1.4e6$. For each group, the worst performance (i.e., the highest notification rate) occurs when both `numStdDevs` and `decayConstant` are equal to 0. This case is equivalent to the static approach. With the exception of the 6-second group shown in Figures 2(a) and 2(d), the notification rate generally decreases as `decayConstant` increases. Again, with the exception of the 6-second group, each group has a trough around `decayConstant` equal to 400,000 where it reaches a local minimum and then increases slightly before permanently decreasing toward 0. Devices reporting at 6-second intervals are virtual machines experiencing very little disruption. This is why devices in this group experience the lowest notification rate, as well as, the least variance in rate with respect to the parameters.

5.3. Static Versus Adaptive

Figure 3 depicts the behavior of both methods on a single device over time. The x-axis represents time, and the y-axis represents the interval between consecutive reports, in seconds. The static approach is considered in Figure 3(a), and the adaptive approach is considered in Figure 3(b). We observe a more tailored fit to the behavior of the device in the adaptive approach. Using the static approach, the majority of devices exhibit a notification rate below 30%, while some generate notification rates of 100%, and a few generate rates of 200%. The highest notification rate for the

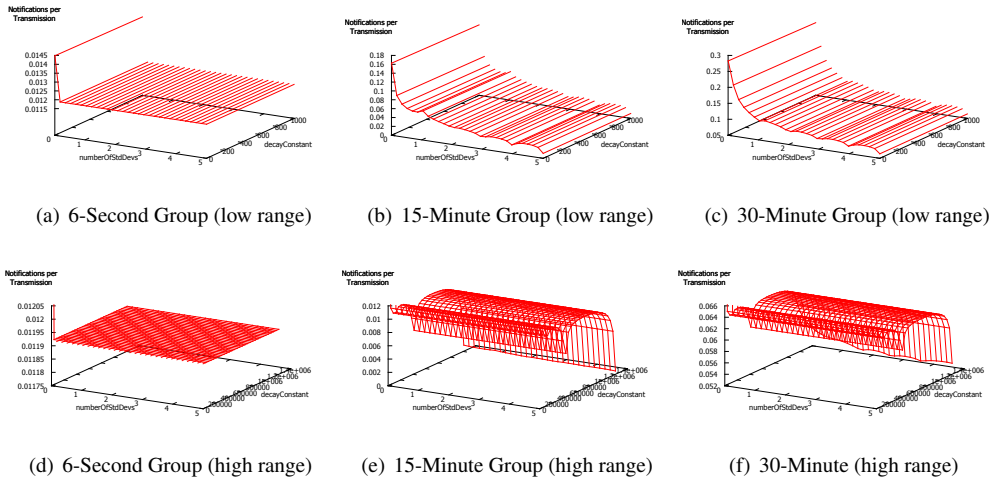


Figure 2. Effect of decayConstant and numOfStdDevs (windowSize=24-hours)

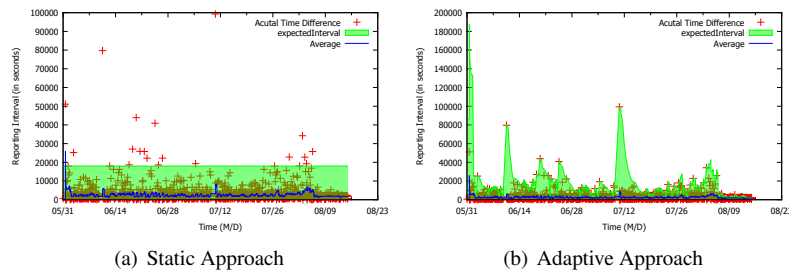


Figure 3. Static vs. Adaptive Approach

adaptive approach is 45%. This shows that in terms of notification rate, the adaptive approach can be used to generate fewer notifications. In some cases, variation may be intolerable, and users may wish to define malfunction using a constant threshold.

6. Conclusion

We presented two approaches to identifying malfunctioning sensors in a streaming monitoring network: a *static* approach and an *adaptive* approach. Trade-offs exist between the two approaches. The static approach may generate too few or too many notifications based on network managers' estimates of reporting behavior. The adaptive approach may generate too many notifications in sensors with sporadic variation. However, we found that both approaches are successful in detecting variations in a sensor's behavior and notifying personnel of sensor failure in real-time. With this method, a sensor cannot be flagged as malfunctioning unless it is actually exhibiting abnormal behavior. Due to decayConstant, it is possible to neglect repeated malfunctions within a close time frame, but this is by design. This work was supported by the NSF (CNS-0745846).

References

- [1] N. Bartzoudis and K. McDonald-Maier. An adaptive processing node architecture for validating sensors reliability in a wind farm. In *BLISS 2007*, pages 83–86, Aug 2007.
- [2] R.N. Duche and N.P. Sarwade. Sensor node failure detection based on round trip delay and paths in wsns. *Sensors Journal, IEEE*, 14(2):455–464, Feb 2014.
- [3] Stephen O. Friend and others. Standard deviation: The new standard for out-of-pattern transaction analysis. *ACAMS Today*, January/February 2009.
- [4] Shuo Guo et al. Find: Faulty node detection for wireless sensor networks. *SenSys '09*, pages 253–266, New York, NY, USA, 2009. ACM.
- [5] MichaelA Hayes and MiriamAM Capretz. Contextual anomaly detection framework for big sensor data. *Journal of Big Data*, 2(1), 2015.
- [6] Chun Lo et al. Pair-wise reference-free fault detection in wireless sensor networks. *IPSN*, pages 117–118. ACM, 2012.
- [7] M.R. Napolitano et al. Kalman filters and neural-network schemes for sensor validation in flight control systems. *Control Systems Technology, IEEE Transactions on*, 6(5):596–611, Sep 1998.
- [8] OASIS. AMQP:advanced message queuing protocol. www.amqp.org/about/what, 2015.
- [9] Pivotal. RabbitMQ: messaging that just works. www.rabbitmq.com, 2014.
- [10] Nithya others Ramanathan. *SenSys '05*, pages 255–267. ACM, 2005.
- [11] Intelligent River. *Intelligentriver®: from observational to operational*. intelligentriver.org, 2015.
- [12] Bilal Shebaro et al. Fine-grained analysis of packet loss symptoms in wireless sensor networks. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, *SenSys '13*, pages 38:1–38:2, New York, NY, USA, 2013. ACM.
- [13] S. Taniguchi and Y. Dote. Sensor fault detection for uninterruptible power supply (ups) control system using fast fuzzy-neural network and immune network. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 1, pages 99–104 vol.1, 2001.
- [14] Ehsan Ullah Warriach et al. Fault detection in wireless sensor networks: A hybrid approach. *IPSN*, pages 87–88, New York, NY, USA, 2012. ACM.
- [15] D.L. White et al. The Intelligent River©: Implementation of sensor web enablement technologies across three tiers of system architecture: Fabric, middle-ware, and application. pages 340–348, May 2010.
- [16] Xi-Liang Zhang et al. Sensor fault diagnosis and location for small and medium-scale wireless sensor networks. In *Natural Computation 2010*, volume 7, pages 3628–3632, Aug 2010.
- [17] J. Zhao et al. Computing aggregates for monitoring wireless sensor networks. In *The IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, pages 139–148, May 2003.

Fault-Based Testing of Combining Algorithms in XACML3.0 Policies

Dianxiang Xu, Ning Shen, Yunpeng Zhang

Department of Computer Science

Boise State University

Boise, ID 83725, USA

{dianxiangxu, ningshen, yunpengzhang}@boisestate.edu

Abstract— With the increasing complexity of software, new access control methods have emerged to deal with attribute-based authorization. As a standard language for attribute-based access control policies, XACML offers a number of rule and policy combining algorithms to meet different needs of policy composition. Due to their variety and complexity, however, it is not uncommon to apply combining algorithms incorrectly, which can lead to unauthorized access or denial of service. To solve this problem, this paper presents a fault-based testing approach for determining incorrect combining algorithms in XACML 3.0 policies. It exploits an efficient constraint solver to generate queries to which a given policy produces different responses than its combining algorithm-based mutants. Such queries can determine whether or not the given combining algorithm is used correctly. Our empirical studies using sizable XACML policies have demonstrated that our approach is effective.

Keywords— *Combining algorithm, constraint solving, fault-based testing, test generation, XACML.*

I. INTRODUCTION

In security-intensive software, access control is a fundamental mechanism for preventing malicious or accidental violation of security requirements by regulating user access to resources. An access control policy defines the conditions under which access to resources can be granted and to whom. Given an access request, it yields an access decision such as permit or deny. With the increasing complexity of software, access control methods have evolved from popular role-based access control to Attribute-Based Access Control (ABAC). ABAC enables fine-grained access control by combining various attributes of authorization elements into access control decisions. These attributes are predefined characteristics of subjects (e.g., job title and age), resources (e.g., data, programs, and networks), actions, and environments (e.g., current time and IP address) [7]. ABAC also facilitates collaborative policy administration within a large enterprise or across multiple organizations. In a large enterprise, for example, elements of authorization policies may be managed by different departments, such as the Information Technology department, Human Resources, the Legal department, and the Finance department [13]. Individual rules or policies are composed into a whole in order to make consistent access decisions.

XACML (eXtensible Access Control Markup Language) [13] is an OASIS standard for specifying ABAC policies in the

XML format. To support flexible policy composition, XACML 3.0 provides 11 rule combining algorithms and 12 policy combining algorithms. A combining algorithm aims at rendering a single access decision by combining the decisions of individual access control rules or policies. Due to the variety of combining algorithms and subtle similarities between the combining algorithms, it is not uncommon to use them incorrectly when XACML3.0 policies are authored. A user may inadvertently select an incorrect combining algorithm or intentionally apply an incorrect combining algorithm due to misunderstanding. Furthermore, for certain rules (or policies), different combining algorithms can be functionally equivalent and result in the same response to every access request. In an evolving process of policy development and maintenance, however, a previously working combining algorithm may become incorrect after new rules or policies are added in a way that implicitly breaks the constraints on functional equivalence. Needless to say, incorrect combining algorithms in XACML policies can lead to devastating consequences, such as unauthorized access and denial of service.

This paper presents a fault-based testing approach for determining existence or absence of incorrect combining algorithms in XACML 3.0 policies. Given an XACML policy (or policy set), our approach analyzes whether the given combining algorithm is functionally equivalent to each of the candidate combining algorithms with respect to the rules in the given policy (or policies in the given policy set). If they can be different, our approach exploits a constraint solver to generate a query to which the two combining algorithms result in different responses. The combining algorithm is correct only if it produces correct responses to such queries. In theory, the query generation involves an NP-hard problem because the targets and conditions in XACML rules, policies and policy sets can be complex first-order logic formulas with user-defined functions. In practice, our case studies have demonstrated that the implementation of our approach based on an efficient constraint solver Z3-str [6][15] is both feasible and effective for dealing with sizable XACML policies.

The remainder of this paper is organized as follows. Section II gives a brief introduction to XACML policies and combining algorithms. Section III describes the fault-based testing approach. Section IV elaborates on fault-based test generation. Section V presents the empirical studies. Section VI reviews related work. Section VII concludes this paper.

II. XACML POLICIES AND COMBINING ALGORITHMS

The main components of the XACML3.0 model are rule, policy, and policy set. A rule consists of a target, a condition, and an effect. The target is a logical expression that specifies the set of requests to which the rule is intended to apply. The condition is a Boolean expression that refines the applicability of the rule established by the target. Predicates in target and condition are defined over attributes and attribute values (e.g., $\text{age} \geq 18$). A policy comprises a policy target, a rule-combining algorithm identifier, and a list of rules. A policy set consists of a policy set target, a policy-combining algorithm identifier, and a list of policies or policy sets. Figure 1 shows the relationships between the main elements of XACML3.0. For simplicity, this paper focuses on policies and rule combining algorithms.

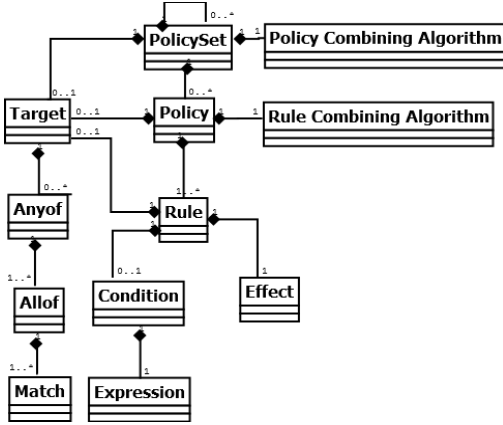


Figure 1. Main language elements of XACML 3.0

Formally, a policy $P = \langle PT, CA, R \rangle$ consists of a policy target PT , a rule combining algorithm CA , and a list of rules R^1 . Each rule $r_i \in R$ is a triple $\langle rt_i, rc_i, re_i \rangle$, where rt_i is the rule's target, rc_i is the rule's condition, and re_i is the rule's effect (either *Deny* or *Permit*). r_i is called a permit rule if $re_i = \text{Permit}$; r_i is called a deny rule if $re_i = \text{Deny}$; rt_i and rc_i are optional. A rule without target and condition, denoted by $\langle _ , _ , re_i \rangle$ is called a default rule.

An access request (also called query) consists of a list of attribute assignments: $\{x_1=V_1, x_2=V_2, \dots\}$, where x_i is an attribute name and V_i is a value assigned to x_i . The decision of rule $r = \langle rt, rc, re \rangle$ with respect to request q , denoted by $d(r, q)$, is defined as follows:

- *Permit*: access is granted when rule effect $re = \text{Permit}$, query q matches policy target PT and rule target rt , and rule condition rc is true with respect to q .
- *Deny*: access is denied when $re = \text{Deny}$, q matches PT and rt , and rc is true with respect to q .
- *N/A*: q is not applicable – q does not match rt or rc evaluate to false with respect to q .
- *I(D)*: An error occurred when rt or rc was evaluated and $re = \text{Deny}$. The decision could have evaluated to *Deny* if no error had occurred.

- *I(P)*: An error occurred when rt or rc was evaluated and $re = \text{Permit}$. The decision could have evaluated to *Permit* if no error had occurred.

For convenience, we use *N/A*, *I(D)*, *I(P)*, and *I(DP)* to denote the following decisions respectively: *NotApplicable*, *Indeterminate {D}*, *Indeterminate {P}*, and *Indeterminate {DP}*. So $d(r, q) \in \{\text{Permit}, \text{N/A}, \text{I(P)}\}$ if r is a permit rule, and $d(r, q) \in \{\text{Deny}, \text{N/A}, \text{I(D)}\}$ if r is a deny rule. For a default rule $r = \langle _ , _ , re \rangle$, $d(r, q) = re$ for any q .

Given query q , rules r_1, r_2, \dots, r_n in policy $P = \langle PT, CA, R \rangle$ may yield different decisions. The rule combining algorithm CA combines the decisions of individual rules into a single policy-level decision, denoted as $d(P, q)$. In XACML 3.0, there are 11 rule combining algorithms. Four are for compatibility support of old versions - *Legacy Ordered-deny-overrides*, *Legacy Permit-overrides*, *Legacy Ordered-permit-overrides*, and *Legacy Ordered-permit-overrides*. In Balana [1] (an open source implementation of XACML3.0 based on which our approach is developed), the implementations of *Ordered-deny-overrides* and *Ordered-permit-overrides* are the same as *Deny-overrides* and *Permit-overrides*. Thus, this paper focuses on five rule combining algorithms: *Deny-overrides*, *Deny-unless-permit*, *Permit-overrides*, *Permit-unless-deny*, and *First-applicable*. Their meanings are as follows:

- *Deny-overrides*: Intended for those cases where a deny decision should have priority over a permit decision;
- *Permit-overrides*: Intended for the cases where a permit decision should have priority over a deny decision.
- *Deny-unless-permit*: Intended for those cases where a permit decision should have priority over a deny decision, and an "Indeterminate" or "NotApplicable" must never be the result.
- *Permit-unless-deny*: Intended for those cases where a deny decision should have priority over a permit decision, and an "Indeterminate" or "NotApplicable" must never be the result.
- *First-applicable*: Rules are evaluated in the order in which they are listed. If a rule's target matches and condition evaluates to "True", then return the rule's effect (*Permit* or *Deny*). If the target or condition evaluates to "False", the next rule is evaluated. If no further rule exists, then return "NotApplicable". If an error occurs, then return "Indeterminate", with the appropriate error status.

Given policy $P = \langle PT, CA, R \rangle$, the set of possible policy decisions depends on CA . For example, *Deny-overrides*, *Permit-overrides*, and *First-applicable* may yield one of the following six decisions: $\{\text{Permit}, \text{Deny}, \text{N/A}, \text{I(D)}, \text{I(P)}, \text{I(DP)}\}$, where *I(DP)* refers to *Indeterminate{DP}*. *I(DP)* results from one of the following situations: (a) an error occurred when policy target PT was evaluated and the decision could have evaluated to *Deny* or *Permit* if no error had occurred; (b) there is a permit rule that evaluates to *I(P)* and a deny rule that evaluates to *I(D)* or *Deny* when $CA = \text{Permit-overrides}$; (c) there is a deny rule that evaluates to *I(D)* and a permit rule that evaluates to *I(P)* or *Permit* when $CA = \text{Deny-overrides}$. *Deny-unless-permit* and *Permit-unless-deny* result in either *Permit* or *Deny*.

¹ In XACML, a policy also has other components, such as obligations and advice. We do not consider these components due to their irrelevance to the research in this paper.

III. FAULT-BASED TESTING OF COMBINING ALGORITHMS

Fault-based testing aims to determine the existence or absence of a hypothesized fault [12]. It has been widely used to generate test cases or evaluate the quality of given tests. This paper focuses on fault-based test generation for incorrect combining algorithm in policy $P = \langle PT, CA, R \rangle$. The basic idea is as follows: assuming CA is faulty and CA' is the correct combining algorithm, the fault-based approach generates a query q such that $d(P, q) \neq d(P', q)$, where $P' = \langle PT, CA', R \rangle$, called P 's mutant. P' has the same policy target and rules as P . According to the correct response to q (called oracle value, denoted as $o(q)$), we can determine whether CA or CA' is faulty. Note that, when testing P , we do not know which combining algorithm is the right one. However, it must be in the given set of rule combining algorithms (denoted as RCA). RCA does not have to contain all the combining algorithms in XACML. It can be a subset, depending on the application. For instance, a meaningful set of combining algorithms to be considered for a particular application might be $\{Permit-overrides, Permit-unless-deny, First-applicable\}$, rather than all the 11 rule combining algorithms in XACML 3.0. As such, our approach considers each possible mutant $P' = \langle PT, CA', R \rangle$ where $CA' \in RCA$ and $CA' \neq CA$ and aims to generate a query to show the difference between P and each P' .

Although CA and CA' are meant to be different, P and P' can be functionally equivalent for certain PT and R , i.e., $d(P, q) = d(P', q)$ for any query q . For example, if R has only permit rules, *Deny-overrides* and *Permit-overrides* would make no difference. Let $query(P, P')$ denote the function that returns *null* if P and P' are functionally equivalent, otherwise returns a query q such that $d(P, q) \neq d(P', q)$. Let $Q = \{q: q = query(P, P') \wedge q \neq null\}$ for each mutant $P' = \langle PT, CA', R \rangle$, $CA' \in RCA$ and $CA' \neq CA$. CA in P is correct if and only if $d(P, q) = o(q)$ for any $q \in Q$. In other words, CA is incorrect if there exists $q \in Q$ such that $d(P, q) \neq o(q)$. Here, determining whether the given combining algorithm is correct or not requires user to define $o(q)$ according to the access control requirements. In our approach, the maximum number of queries for which user needs to define oracle values is $|RCA| - 1$. This is much more effective than reviewing all the rules in the policy or testing the policy with many queries. As reviewed in Section VI, the existing testing methods for XACML policies do not target the detection of incorrect combining algorithms. They all generate a large number of queries to which user has to define the oracle value of each query.

The fault-based testing of XACML combining algorithms in our approach involves two issues: (1) determine when P and P' are functionally equivalent with respect to the given policy target and rules; and (2) when P and P' are not functionally equivalent, find a query q such that $d(P, q) \neq d(P', q)$. To address the first issue, our technical report [14] has formalized the semantic differences between the five rule combining algorithms and between the six policy combining algorithms with 49 theorems. These theorems describe the necessary and sufficient conditions under which different combining algorithms are functionally equivalent. Based on [14], this paper focuses on the second issue by exploiting a constraint solver for automated test generation. For example, the following two theorems capture the semantic difference

between rule combining algorithms *Deny-overrides* and *Permit-overrides*. Detailed proofs can be found in [14].

Theorem 1. Given policy $P = \langle PT, Deny-overrides, R \rangle$ and $P' = \langle PT, Permit-overrides, R \rangle$. If r_i ($1 \leq i \leq n$) are all permit rules or r_i ($1 \leq i \leq n$) are all deny rules, then P and P' are functionally equivalent.

Theorem 2. Given policy $P = \langle PT, Deny-overrides, R \rangle$ and $P' = \langle PT, Permit-overrides, R \rangle$, where R has at least one permit rule and at least one deny rule. For any q , $d(P, q) \neq d(P', q)$ if and only if there exists permit rule $r_i = \langle rt_i, rc_i, Permit \rangle \in R$, deny rule $r_j = \langle rt_j, rc_j, Deny \rangle \in R$, and query q , such that:

- (a) $d(r_i, q) = Permit \wedge d(r_j, q) \in \{Deny, I(D)\}$ or
- (b) $d(r_i, q) = I(P) \wedge d(r_j, q) = Deny$.

The above theorems lay the foundation for generating query q such that $d(P, q) \neq d(P', q)$. The corresponding test generation algorithm is described in the next section.

IV. FAULT-BASED TEST GENERATION

This section discusses how to design and implement *query* (P, P') using constraint solver Z3-str. Z3 [6] is an efficient SMT (Satisfiability Modulo Theories) Solver from Microsoft Research. SMT generalizes Boolean Satisfiability (SAT) by adding equality reasoning, arithmetic, fixed-size bit-vectors, arrays, quantifiers, and other useful first-order theories. Z3 supports basic data types (e.g., *Int* and *Bool*) as well as data structures (e.g., *Array*, *List*, *BitVec*, and *Records*). However, Z3 does not directly deal with strings. To address this issue, Z3-str [15] extends Z3 by treating strings as a primitive type and supporting common string operations.

In the following, we first introduce the basic functions that generate queries for a pair of rules and then describes how they are used in the query generation algorithms for comparing combining algorithms. These basic functions represent the queries used in the detailed proofs of the theorems [14]. We also discuss how to implement the basic query generation functions by transforming the corresponding targets and conditions of an XACML policy into the input of Z3-str.

A. Query Generation Functions

Suppose $r_1 = \langle rt_1, rc_1, re_1 \rangle$ and $r_2 = \langle rt_2, rc_2, re_2 \rangle$ are two rules. E, N , and I stand for *Effect* (*Permit* or *Deny*), *N/A*, and *Indeterminate*, respectively. For simplicity, here we do not consider targets of policies or policy sets, which are handled similarly. The basic query generation functions are as follows:

- $queryE_E(r_1, r_2)$: generate a query q to make both r_1 and r_2 produce the specified effects re_1 and re_2 , respectively (i.e., $d(r_1, q) = re_1$ and $d(r_2, q) = re_2$). In this case, the rule targets and conditions are all satisfied, i.e., $rt_1 \wedge rc_1 \wedge rt_2 \wedge rc_2$.
- $queryE_N(r_1, r_2)$: generate a query q to make r_1 produce the specified effect re_1 and r_2 produce *N/A* (i.e., $d(r_1, q) = re_1$ and $d(r_2, q) = N/A$). In this case, $rt_1 \wedge rc_1 \wedge \neg(rt_2 \wedge rc_2)$.
- $queryE_I(r_1, r_2)$: generate a query q to make r_1 produce the specified effect re_1 and r_2 produce *Indeterminate*

(i.e., an error in the process of evaluation). $d(r_1, q) = re_1$ and $d(r_2, q) = I(D)$ when $re_2 = Deny$ or $I(P)$ when $re_2 = Permit$.

- $queryI_N(r_1, r_2)$: generate a query q to make r_1 produce *Indeterminate* and r_2 produce *N/A*. In this case, $d(r_1, q) = I(D)$ when $re_1 = Deny$ or $I(P)$ when $re_1 = Permit$. $d(r_2, q) = N/A$.
- $queryN_N(r_1, r_2)$: generate a query q to make both r_1 and r_2 produce *N/A* (i.e., $d(r_1, q) = N/A$, $d(r_2, q) = N/A$). In this case, $\neg (rt_1 \wedge rc_1) \wedge \neg (rt_2 \wedge rc_2)$.
- $queryI_I(r_1, r_2)$: generate a query to make both r_1 and r_2 produce *Indeterminate*.

Using the above functions, we can formalize the algorithms for each pair of the combining algorithms according to the formalized semantic difference [14]. Consider *Deny-overrides* and *Permit-overrides* as an example. Algorithm 1 below describes the query generation process based on Theorems 1 and 2. According to Theorem 1, if the rules are all permit rules or all deny rules, they are functionally equivalent and thus no query can be generated. This is corresponding to lines 1-4 in Algorithm 1. According to Theorem 2, if a query makes a pair of permit and deny rules produce *Permit* and *Deny* (or *I(D)*) respectively (i.e., condition (a) in Theorem 2), then *Deny-overrides* and *Permit-overrides* produce different responses to this query. This is corresponding to lines 6-18 in Algorithm 1. Similarly, if a query makes a pair of deny and permit rules produce *Deny* and *I(P)* respectively (i.e., condition (b) in Theorem 2), *Deny-overrides* and *Permit-overrides* also produce different responses to this query. This is done by lines 19-26 in Algorithm 1.

Algorithm 1: $query(P = \langle PT, Deny-overrides, R \rangle, P' = \langle PT, Permit-overrides, R \rangle)$

Function: generate q such that $d(P, q) \neq d(P', q)$ if feasible.

Input: $P = \langle PT, Deny-overrides, R \rangle, P' = \langle PT, Permit-overrides, R \rangle$

Output: query q or null

1. if $re_i = Permit$ for all i ($1 \leq i \leq n$) // Theorem 1
2. return null;
3. else if $re_i = Deny$ for all i ($1 \leq i \leq n$) // Theorem 1
4. return null;
5. else // Theorem 2
6. for $r_i = 1$ st permit rule to last permit rule, do
7. for $r_j = 1$ st deny rule to last deny rule, do:
8. $q = queryE_E(r_i, r_j)$;
9. if $q \neq null$
10. return q ;
11. else
12. $q = queryE_I(r_i, r_j)$;
13. if $q \neq null$
14. return q ;
15. end if
16. end if
17. end for
18. end for // condition (a)
19. for $r_i = 1$ st deny rule to last deny rule, do:
20. for $r_j = 1$ st permit rule to last permit rule, do:
21. $q = queryE_I(r_i, r_j)$;
22. if $q \neq null$
23. return q ;

24. end if
25. end for
26. end for // condition (b)
27. return null;
28. end if

B. Transforming XACML Constructs to Z3-str

The aforementioned basic query generation functions are realized by transforming XACML constructs (i.e., targets and conditions) to the input of Z3-str, executing Z3-str with the transformed input, and translating the result of Z3-str to an XACML query. Converting XACML targets and conditions consists of two steps. In the first step, attributes in the given targets and conditions (i.e., rt_1 , rc_1 , rt_2 , and rc_2 in the aforementioned basic query generation functions) are defined as typed variables in Z3-str. The attributes have to be renamed in Z3-str because the syntax of identifiers is different. The data type of each XACML attribute is also changed to a data type in Z3-str. XACML3.0 has 17 basic data types: *string*, *Boolean*, *integer*, *double*, *time*, *date*, *dateTime*, *anyURI*, *hexBinary*, *base64 Binary*, *dayTimeDuration*, *yearMonthDuration*, *rfc822Name*, *x500Name*, *xpathExpression*, *ipAddress*, and *dnsName*. Each of these data types can be mapped to a basic data type or data structure in Z3-str. For example, *date* in XACML can be corresponding to a record with three integer fields. In the second step, the logical expressions of targets and conditions are converted into Z3-str expressions. As the conversion involves many non-trivial details, here we use some examples to illustrate the idea. Consider the following rule target in XACML (for clarity, URI links are omitted):

```
<AnyOf>
  <AllOf>
    <Match MatchId="...function:string-equal">
      <AttributeValue DataType="...string">book</AttributeValue>
      <AttributeDesignator AttributeId="...resource:resource-id"
        Category="...attribute-category:resource"
        DataType="...string" MustBePresent="true"/>
    </AttributeDesignator>
    </Match>
    <Match MatchId="...function:string-equal">
      <AttributeValue DataType="...string">buy</AttributeValue>
      <AttributeDesignator AttributeId="...action:action-id"
        Category="...attribute-category:action"
        DataType="...string" MustBePresent="true"/>
    </AttributeDesignator>
    </Match>
  </AllOf>
</AnyOf>
<AnyOf>
  <AllOf>
    <Match MatchId="...function:string-equal">
      <AttributeValue DataType="...string">teacher</AttributeValue>
      <AttributeDesignator AttributeId="...subject:subject-id"
        Category="...subject-category:access-subject"
        DataType="...string" MustBePresent="true"/>
    </AttributeDesignator>
    </Match>
  </AllOf>
</AnyOf>
<AnyOf>
  <AllOf>
    <Match MatchId="...function:string-equal">
      <AttributeValue DataType="...string">workday</AttributeValue>
      <AttributeDesignator AttributeId="...environment:day"
        Category="...environment-category:environment"
        DataType="...string" MustBePresent="true"/>
    </AttributeDesignator>
  </AllOf>
</AnyOf>
```

```

</Match>
</AllOf>
</AnyOf>

```

The above target has the same meaning as the following logic formula:

```

((resource-id = book ∧ action-id = buy)
 ∨ subject-id = teacher) ∧ (day=workday)

```

where attributes resource-id, action-id, subject-id, and day are all of the string type. A non-error query should provide a value for each attribute because of *MustBePresent*="true". To generate a query to satisfy the target condition, it can be converted into the following Z3-str input:

```

(declare-variable resourceid String)
(declare-variable actionid String)
(declare-variable subjectid String)
(declare-variable day String)
(assert (and (or (and (=resourceid "book") (= actionid
"buy"))) (and (=subjectid "teacher")))) (or (and (= day
"workday"))))
(check-sat)
(get-model)

```

The “declare-variable” statements define variables for the attributes, and the “assert” expression describes the constraint to be solved.

For query generation functions $queryE_E(r_1, r_2)$, $queryE_N(r_1, r_2)$, $queryN_N(r_1, r_2)$, we only need to make the targets and conditions true or false (e.g., $rt_1 \wedge rc_1 \wedge rt_2 \wedge rc_2$ for $queryE_E(r_1, r_2)$). The other functions, $queryE_I(r_1, r_2)$, $queryN_I(r_1, r_2)$, and $queryI_I(r_1, r_2)$, however, generate queries to produce *Indeterminate* by triggering an error status. Generation of such queries is much more complicated as discussed below. Typically, such a query should make part of a target (or condition) produce an error while ensuring the other part to evaluate to true or false. Therefore, query generation may involve selecting an appropriate attribute to trigger an error. In the above example, if we choose attribute *day* to trigger an error (e.g., a query that provides no value for *day*), then we have to ensure the resultant query must satisfy the following condition:

```

((resource-id=book ∧ action-id = buy) ∨ subject-id = teacher)

```

If a query does not meet this condition, then *day=workday* will not be evaluated. Thus, it will not produce an error. If we choose *subject-id* to produce an error, then the resultant query should make $(resource-id = book \wedge action-id = buy)$ evaluate to false, otherwise *subject-id = teacher* will not be evaluated.

Generally, there are a great variety of errors that can result in a response of *Indeterminate* in XACML 3.0 [12]. The errors can be caused by problematic policies, queries, or both. Here our focus is on the errors caused by queries, assuming that the given policy is well-defined except for incorrect combining algorithm. In addition, $queryE_I(r_1, r_2)$, $queryN_I(r_1, r_2)$, and $queryI_I(r_1, r_2)$ need to consider interactions of attributes in both rules. When both rules use the same set of attributes, it may be infeasible to create a particular type of error to obtain *Indeterminate*. This is because a query making one rule evaluate to $I(D)$ or $I(P)$ may also make the other rule evaluate to $I(D)$ or $I(P)$.

V. EMPIRICAL STUDIES

We have implemented our approach based on Balana [1] and applied it to nine case studies with different levels of complexity. The case studies are summarized in Table I. K-market is a sample application of Balana with a total of 12 rules in three policies. It is the only one that is originally encoded in XACML 3.0. *itrust*, *pluto*, *conference*, and *fedora* are real-world policies from literature. They were originally encoded in XACML 2.0 or 1.0. In this paper, we manually converted them into XACML 3.0 with the same semantics. *itrustX* ($X=5, 10, 20$, or 40) is a policy synthesized from *itrust*. It has X times as many rules as *itrust*. The new rules in *itrustX* are created by replicating the existing rules with new attribute values. Because the real-world policies from literature have a small number of rules, we use *itrustX* to evaluate whether or not our approach is applicable to large-scale policies.

TABLE I. SUBJECT POLICIES OF EMPIRICAL STUDIES

Name	#Rules	Combining algorithm	Equivalent combining algorithms
K-market [1]	12	Deny-overrides	None
itrust ²	64	First-applicable	Permit-overrides/ Deny-overrides
pluto	21	Permit-overrides	None
conference	15	Permit-overrides	None
fedora ³	12	Deny-overrides	None
itrust5	320	First-applicable	Permit-overrides/ Deny-overrides
itrust10	640	First-applicable	Permit-overrides/ Deny-overrides
itrust20	1,280	First-applicable	Permit-overrides/ Deny-overrides
itrust40	2,560	First-applicable	Permit-overrides/ Deny-overrides

We treat the combining algorithm in each original policy as the correct one and inspect each policy to determine which combining algorithms are functionally equivalent and which are non-equivalent for each given policy. As shown in Table I, the policies in *itrust* and its variations have equivalent algorithms. As the correct combining algorithms in the given policies are already assumed, the goal of our evaluation is to demonstrate whether or not our approach can detect incorrect combining algorithms and functionally equivalent combining algorithms. Let P_0 and CA_0 denote the correct policy (or policy set) and original combining algorithm respectively. We used the following protocol to conduct the experiment:

- Use the correct policy P_0 to create a policy or policy set P with a different combining algorithm CA (i.e., $CA \neq CA_0$);
- Apply our approach to P , comparing CA to each of the other combining algorithms (including CA_0) and try to generate a query for each pair;
- If no query is generated for $\langle P, P_0 \rangle$ and $d(P, q) = d(P_0, q)$ for each query q generated in the above step, then CA is correct and functionally equivalent to CA_0 , otherwise CA is incorrect.

² <http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=start>

³ <http://www.fedora.info>

The results of our experiments have shown that our approach was able to identify all correct and equivalent combining algorithms as defined in Table I. Consider *itrust* (or *itrustX*). *First-applicable*, *Deny-overrides*, and *Permit-overrides* are equivalent. When any two of them were compared, no query was generated, which means they have no difference. When one of them was compared to *Deny-unless-permit* or *Permit-unless-deny*, however, a query was generated, which means they are different. In *K-market*, *pluto*, *conference*, or *fedora*, a query was generated for each pair of combining algorithms. This means that all the combining algorithms are different with respect to the given policy target and rule.

VI. RELATED WORK

In Cirg [9], tests are generated from counterexamples produced by the change-impact analysis of two synthesized versions. The difference of the two versions of a policy targets a test coverage goal (e.g., rule, or condition). Targen [10] is a test generator for XACML policies that derives access requests to satisfy all the possible combinations of truth values of the attribute id-value pairs found in a given policy. Access requests generated by Cirg and Targen typically use a limited number of subject, resource, action, and environment attributes. A real request, however, could use any combination of attributes. Because requests are encoded in XML, they must conform to the XML Context Schema. To address this issue, Bertolino et al., have developed different test generation algorithms by considering the structures of the Context Schema [2][3][5]. These algorithms can generate requests that use more than one subject, resource, action, or environment attribute. They can also produce robustness tests, where invalid attribute values are generated randomly.

Li et al. have applied symbolic execution technique to generation of access requests for testing XACML policies [8]. They convert the policy under test into semantically equivalent C Code Representation (CCR) and symbolically execute CCR to create test inputs and translate the test inputs to access control requests. Mutation of the XACML policies [4][11] has been commonly used to evaluate the above testing methods. In this paper, however, we use combining algorithm-based mutants to generate queries for determining whether or not the given combining algorithm is correct.

VII. CONCLUSIONS

We have presented the fault-based approach to automated test generation for determining existence or absence of incorrect combining algorithms in XACML3.0 policies. Based on the formalized semantic differences between combining algorithms, our approach exploits a constraint solver to generate a query to show the difference between the given combining algorithms and each of the mutants. Our case studies have demonstrated that the approach is effective and applicable to sizeable policies. As a byproduct, our approach can be a useful tutoring tool for learning about XACML combining algorithms and their essential differences. When a user is uncertain about which combining algorithm should be used, she may compare similar algorithms and generate requests to show the difference. This will help the user get an

accurate understanding and choose the right combining algorithm.

This paper offers a first step towards general fault-based testing of XACML policies. Incorrect combining algorithms are just one type of faults in XACML policies. Other fault types include incorrect (policy set, policy, and rule) target, incorrect rule effect, and incorrect rule conditions [4][11]. Our future work will investigate fault-based test generation algorithms for each of these uncovered fault types.

ACKNOWLEDGMENT

This work was supported in part by US National Science Foundation (NSF) under grants CNS 1123220 and 1359590.

REFERENCES

- [1] Balana, "Open source XACML 3.0 implementation," <http://xacmlinfo.org/2012/08/16/balana-the-open-source-xacml-3-0-implementation/>, 2012.
- [2] A. Bertolino, S. Daoudagh, F. Lonetti, and E. Marchetti. "Automatic XACML requests generation for policy testing." Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST), 2012, pp.842-849.
- [3] A. Bertolino, S. Daoudagh, F. Lonetti, and E. Marchetti. "The X-CREATE Framework-A Comparison of XACML Policy Testing Strategies." *Proc. of the 8th International Conference on Web Information Systems and Technologies (WEBIST)*. pp.155-160.
- [4] A. Bertolino, S. Daoudagh, F. Lonetti, and E. Marchetti. "Xacmut: Xacml 2.0 mutants generator." Sixth IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2013, pp.28-33.
- [5] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti and L. Schilders. "Automated testing of extensible access control markup language-based access control systems." *Software, IET 7.4* (2013), pp.203-212.
- [6] L. De Moura, and N. Bjørner. "Z3: An efficient SMT solver." *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, 2008, pp.337-340.
- [7] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnizer, K. Sandlin, R. Miller and K. Scarfone. "Guide to Attribute Based Access Control (ABAC) Definition and Considerations." NIST Special Pub 800 (2014): 162.
- [8] Y. C. Li, Y. Li, L. Z. Wang, and G. Chen. "Automatic XACML Requests Generation for Testing Access Control Policies." *Proc. of the 26th International Conf. on Software Engineering and Knowledge Engineering (SEKE'14)*, Vancouver, July 2014.
- [9] E. Martin and T. Xie. "Automated test generation for access control policies," in *Supplemental Proc. of ISSRE*, November 2006.
- [10] E. Martin, and T. Xie. "Automated test generation for access control policies via change-impact analysis." *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*. IEEE Computer Society, 2007, pp.5-11.
- [11] E. Martin, and T. Xie. "A fault model and mutation testing of access control policies." *Proceedings of the 16th International Conference on World Wide Web*. ACM, 2007, pp.667-676.
- [12] L. J. Morell. "A theory of fault-based testing". *IEEE Trans. on Software Engineering*, Vol. 16, no.8, August 1990, pp. 844-857.
- [13] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," <http://www.oasisopen.org/committees/xacml/>. 2013.
- [14] D. Xu, Y. Zhang, N. Shen. "Formalizing semantic differences of combining algorithms in XACML 3.0," Technical Report, Boise State University. <http://cs.boisestate.edu/~dxu/research/TR-BSU-CS-SEAL2014-001.pdf>
- [15] Y. Zheng, X. Zhang, and V. Ganesh, "Z3-str: A Z3-based string solver for web application analysis," *Proc. of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13)*, pp.114-124.

Combining Feature Subset Selection and Data Sampling for Coping with Highly Imbalanced Software Data

Kehan Gao

Eastern Connecticut State University
Willimantic, Connecticut 06226
gaok@easternct.edu

Taghi M. Khoshgoftaar & Amri Napolitano

Florida Atlantic University
Boca Raton, Florida 33431
khoshgof@fau.edu, amrifau@gmail.com

Abstract—In the software quality modeling process, many practitioners often ignore problems such as high dimensionality and class imbalance that exist in data repositories. They directly use the available set of software metrics to build classification models without regard to the condition of the underlying software measurement data, leading to a decline in prediction performance and extension of training time. In this study, we propose an approach, in which feature selection is combined with data sampling, to overcome these problems. Feature selection is a process of choosing a subset of relevant features so that the quality of prediction models can be maintained or improved. Data sampling seeks a more balanced dataset through the addition or removal of instances. Three different approaches would be produced when combining these two techniques: 1- sampling performed prior to feature selection, but retaining the unsampled data instances; 2- sampling performed prior to feature selection, retaining the sampled data instances; 3- sampling performed after feature selection. The empirical study was carried out on six datasets from a real-world software system. We employed one filter-based (no learning algorithm involved in the selection process) feature subset selection technique called correlation-based feature selection combined with the random undersampling method. The results demonstrate that sampling performed prior to feature selection, but retaining the unsampled data instances (Approach 1) performs better than the other two approaches.

Index Terms—software defect prediction, feature selection, data sampling, subset selection

I. INTRODUCTION

Quality and reliability are the most important factors that determine success or failure of software projects, especially for high-assurance and mission-critical systems. Early detection of faults prior to system deployment and operation can help for reducing development costs and allowing for timely improvement to the software product. Various techniques have been developed for this purpose, and some of them have achieved beneficial results. One such technique is software quality classification, in which a classifier is constructed on historical software data (including software metrics and fault data) collected during the software development process, then that classifier is used to classify new program modules under development as either fault-prone (*fp*) or not-fault-prone (*nfp*) [1]. This prediction can help practitioners to identify potentially problematic modules and assign project resources

accordingly. However, two problems, high dimensionality and class imbalance, may affect the classifier's performance.

In the software quality modeling process, high dimensionality occurs when a data repository contains many metrics (features) that are either redundant or irrelevant to the class attribute. Redundant features refer to those having information which is already contained in other features, while irrelevant features are features with no useful information related to the class variable. Several problems may arise due to high dimensionality, including high computational cost and memory usage, a decline in prediction performance, and difficulty of understanding and interpreting the model.

Feature selection is a process of selecting a subset of relevant features for use in model construction, so that prediction performance will be improved or maintained, while learning time is significantly reduced. Feature selection techniques can be categorized as either wrappers or filters based on whether a learning algorithm is involved in the selection process, or be classified into feature subset selection and feature ranking depending on whether features are assessed collectively or individually [2]. Feature ranking scores the attributes based on their individual predictive power. A potential problem of feature ranking is that it neglects the possibility that a given attribute may have better predictability when combined with some other attributes, as compared to when used by itself. Feature subset selection that evaluates a subset of features as a group for suitability can overcome this problem. Wrappers evaluate each subset through a learning algorithm, while filters use a simpler statistical measure or some intrinsic characteristic to evaluate each subset rather than using a learning algorithm. Unfortunately, the building of the classifiers required for wrapper-based feature selection are frequently computationally infeasible. Thus, filter-based subset selection is a promising option as it evaluates subsets but is relatively faster than wrapper-based methods. In this study, we would like to examine one filter-based feature subset selection technique called correlation-based feature selection [3] in the context of software quality modeling.

In addition to an excess number of features, many real-world software datasets have the class imbalance problem,

wherein *nfp* modules significantly outnumber *fp* modules (the class of interest). When training data is imbalanced, traditional machine learning algorithms may have difficulty distinguishing between instances of the two classes. In this scenario, they tend to classify the *fp* modules as *nfp* modules to increase overall prediction accuracy. However, these models are rarely useful, because in software engineering practice, accurately detecting the few faulty modules is of utmost importance at the final stage of system testing, as it can avoid defective software in deployment and operation. Many solutions have been proposed to address the class imbalance problem. A frequently used method is data sampling [4], which attempts to achieve a certain balance (ratio) between the two classes by adding instances to (oversampling), or removing instances from (undersampling), the dataset. In this work, we employ a simple and effective sampling technique, random undersampling.

To cope with both high dimensionality and class imbalance, we proposed a data pre-processing technique in which feature selection is combined with data sampling. Some questions may arise when we combine the two techniques, such as which activity, feature selection or sampling, should be performed first? In addition, given the subset of selected features, should the training data be formed based on the sampled dataset or unsampled dataset? To answer all these questions, we investigate three different approaches: 1- data sampling performed prior to feature selection and the training data formed using selected features along with unsampled data; 2- data sampling performed prior to feature selection and the training data formed using selected features along with sampled data; and 3- data sampling performed after feature selection. In this study, we are interested in learning the impact of the feature subset selection technique on classification results when used along with a sampling method as well as the effects of three approaches on classification performance. To our knowledge, no study have been done for combining a filter-based feature subset selection method with data sampling and investigating the three approaches in the domain of software quality engineering.

The empirical study was carried out over two groups of datasets (each group having three datasets) from a real-world software system, all of which exhibit a high degree of class imbalance between the *fp* and *nfp* classes. Five different learners were used to build classification models. The experimental results demonstrate that data sampling performed prior to feature selection and the training data formed using selected features along with unsampled data (Approach 1) had significantly better performance than sampling performed after feature selection (Approach 3), or retaining the sampled data (Approach 2). As to the classification algorithms, Support Vector Machine presented the best (or close to the best) performance irrespective of training data or approach adopted, and therefore was recommended. Multilayer Perceptron and K Nearest Neighbors showed moderate performance, followed Naïve Bayes. Logistic Regression had fluctuate performance with respect to various approaches used.

The rest of the paper is organized as follows. Section II discusses related work. Section III provides methodology, including more detailed information about feature subset selection, data sampling, three combination approaches, learners, performance metric, and cross-validation applied in this work. A case study is described in Section IV. Finally, the conclusion and future work are summarized in Section V.

II. RELATED WORK

Feature selection is an effective technique to solve the high dimensionality problem, and therefore has been significantly researched. Liu et al. [2] provided a comprehensive survey of feature selection and reviewed its developments with the growth of data mining. At present, feature selection has been widely applied in a range of fields, such as text categorization, remote sensing, intrusion detection, genomic analysis, and image retrieval [5]. Hall and Holmes [6] investigated six attribute selection techniques (information gain, ReliefF, principal components analysis, correlation-based feature selection (CFS), consistency-based subset evaluation (CNS), and wrapper subset evaluation) and applied them to 15 datasets. The comparison results show no single best approach for all situations. However, a wrapper-based approach is the best overall attribute selection schema in terms of accuracy if speed of execution is not a considered factor. Otherwise, CFS, CNS, and ReliefF are overall good performers. Feature selection also gets more attention in the software quality assurance domain [7]. Akalya et al. [8] proposed a hybrid feature selection model that combines wrapper and filter methods and applied it to NASA's public KCI dataset obtained from the NASA IV&V Facility Metrics Data Program (MDP) data repository.

Besides an excess number of attributes, many real-world classification datasets suffer from the class imbalance problem. A considerable amount of research has been done to investigate this problem. Weiss [4] provided a survey of the class imbalance problem and techniques for reducing the negative impact imbalance has on classification performance. An important technique discussed for alleviating the problem of class imbalance is data sampling. The simplest form of sampling is random sampling. Besides that, several more intelligent algorithms for sampling data have been proposed, such as SMOTE [9] and Wilson's Editing [10].

While a great deal of work has been done for feature selection and data sampling separately, limited research has been done and reported on both together, especially in the context of software quality assurance. Among the few studies, Wahono et al. [11] proposed the combination of genetic algorithms with the bagging (bootstrap aggregation) technique for improving the performance of software defect prediction. Genetic algorithms were applied to deal with the feature selection, and bagging was employed to deal with the class imbalance problem. In one of our previous studies [12], we investigated combining feature ranking techniques with data sampling and also examined different combination scenarios. That previous study was focused on feature ranking, while the present research concentrates on feature subset selection.

III. METHODOLOGY

A. Feature Subset Selection

For any feature subset selection method, a key issue discussed is the search strategy which determines how the subsets are generated in the first place in order to avoid the $O(2^n)$ models built with exhaustive search. We use the Greedy Stepwise search mechanism in this paper. Greedy Stepwise starts with an empty working feature set and progressively add features, one at a time, until a stopping criterion is reached. Greedy Stepwise uses forward selection to build the full feature subset starting from the empty set. At each point in the process, the algorithm creates a new family of potential feature subsets by adding every feature (one at a time) to the current best-known set. The merit of all these sets are evaluated, and whichever performs best is the new known best set. This process is repeated until none of the new subsets improve performance. The final new “known-best” subset (that is, the last subset which improved performance over its predecessor) is then given as the procedure’s output.

The main goal of feature selection is to select a subset of features that minimizes the prediction errors of classifiers. In this study, we employ correlation-based subset selection algorithm [3].

The correlation-based algorithm uses the Pearson correlation coefficient [3], which can be calculated using the following formula:

$$M_S = \frac{k\bar{r}_{cf}}{\sqrt{k + k(k-1)\bar{r}_{ff}}}$$

In this formula, M_S is the merit of the current subset of features, k is the number of features, \bar{r}_{cf} is the mean of the correlations between each feature and the class variable, and \bar{r}_{ff} is the mean of the pairwise correlations between every two features.

B. Data Sampling

A variety of data sampling techniques have been studied in the literature, including both majority undersampling and minority oversampling techniques [9], [13]. We employ random undersampling as the data sampling technique in this study. Random undersampling is a simple, yet effective, data sampling technique that achieves more balance in a given dataset by randomly removing instances from the majority (*nfp*) class. The post-sampling class ratio (between *fp* and *nfp* modules) was set to 50:50 throughout the experiments.

C. Three Combination Approaches

The primary goal of this study is to evaluate the data pre-processing technique in which the correlation-based feature subset selection technique is combined with random undersampling. Three different scenarios (also called approaches) would be produced depending on whether sampling is performed before or after feature selection and which dataset, sampled or unsampled data, is used to build a classifier. The three different approaches are described as follows:

- Approach 1: sampling then feature selection retaining the unsampled data instances
- Approach 2: sampling then feature selection retaining the sampled data instances
- Approach 3: feature selection then sampling

Fig. 1 shows the three approaches denoted as DS-FS-UnSam, DS-FS-Sam, and FS-DS, respectively.

D. Learners

The software defect prediction models in this study are built using five different classification algorithms, including Naïve Bayes (NB) [14], MultiLayer Perceptron (MLP) [14], K Nearest Neighbors (KNN) [14], Support Vector Machine (SVM) [15], and Logistic Regression (LR) [14]. Due to space limitations, we refer interested readers to these references to understand how these commonly-used learners function. The WEKA machine learning tool is used to instantiate the different classifiers. Generally, the default parameter settings for the different learners are used (for NB and LR), except for the below-mentioned changes. A preliminary investigation in the context of this study indicated that the modified parameter settings are appropriate.

In the case of MLP, the `hiddenLayers` parameter was changed to ‘3’ to define a network with one hidden layer containing three nodes, and the `validationSetSize` parameter was changed to ‘10’ to cause the classifier to leave 10% of the training data aside for use as a validation set to determine when to stop the iterative training process. For the KNN learner, the `distanceWeighting` parameter was set to ‘Weight by 1/distance’, the `kNN` parameter was set to ‘5’, and the `crossValidate` parameter was turned on (set to ‘true’). In the case of SVM, two changes were made: the complexity constant `c` was set to ‘5.0’, and `build Logistic Models` was set to ‘true’. A linear kernel was used by default.

E. Performance Metric

The Area Under the ROC (receiver operating characteristic) curve (i.e., AUC) is one of the most widely used single numeric measures that provides a general idea of the predictive potential of the classifier. The ROC curve graphs true positive rates versus the false positive rates. Traditional performance metrics for classifier evaluation consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. A classifier that provides a large area under the curve is preferable over a classifier with a smaller area under the curve. A perfect classifier provides an AUC that equals 1. AUC is of lower variance and is more reliable than other performance metrics such as precision, recall, and F-measure [16].

F. Cross-Validation

For all experiments, we employed 10 runs of 5-fold cross-validation (CV). That is, for each run the data was randomly divided into five folds, one of which was used as the test data while the other four folds were used as training data. All the

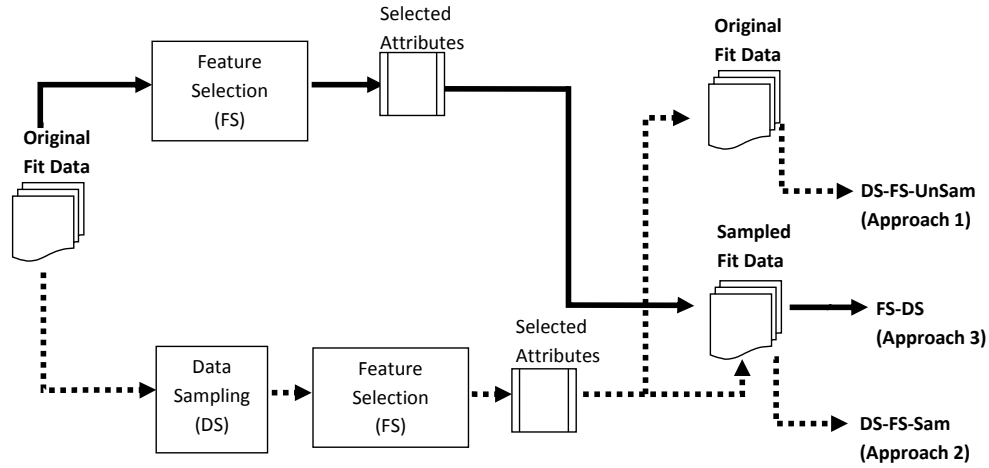


Fig. 1. Three approaches for combining feature selection with data sampling

TABLE I
DATA CHARACTERISTICS

Dataset	Rel.	thd	#Attr.	#Inst.	<i>fp</i>		<i>nfp</i>	
					#	%	#	%
Eclipse 1	2.0	10	209	377	23	6.1	354	93.9
	2.1	5	209	434	34	7.8	400	92.2
	3.0	10	209	661	41	6.2	620	93.8
Eclipse 2	2.0	5	209	377	52	13.8	325	86.2
	2.1	4	209	434	50	11.5	384	88.5
	3.0	5	209	661	98	14.8	563	85.2

preprocessing steps (feature selection and data sampling) were done on the training dataset. The processed training data was then used to build the classification model and the resulting model was applied to the test fold. This cross-validation was repeated five times, with each fold used exactly once as the test data. The five results from the five folds then was averaged to produce a single estimation. In order to lower the variance of the CV result, we repeated the CV with new random splits 10 times. The final estimation is the average results over the 10 runs of 5-fold CV.

IV. A CASE STUDY

A. Datasets

In our experiments, we use publicly available data, namely the Eclipse defect counts and complexity metrics dataset obtained from the PROMISE data repository (<http://promisedata.org>). In particular, we use the metrics and defects data at the software packages level. The original data for Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0 respectively. Each release as reported by Zimmerman et al. [17] contains the following information: the name of the package for which the metrics are collected (name), the number of defects reported six months prior to release (pre-release defects), the number of defects reported six months after release (post-release defects), a set of complexity metrics computed for classes or methods and aggregated in terms of average, maximum, and total (complexity metrics), and the

abstract syntax tree of the package consisting of the node size, type, and frequency (structure of abstract syntax tree(s)). For our study we transform the original data by: (1) removing all non-numeric attributes, including the package names, and (2) converting the post-release defects attribute to a binary class attribute with fault-prone (*fp*) being the minority class and not-fault-prone (*nfp*), the majority class. Membership in each class is determined by a post-release defects threshold *thd*, which separates *fp* from *nfp* packages by classifying packages with *thd* or more post-release defects as *fp* and the remaining as *nfp*. In our study, we use $thd = \{10, 5\}$ for releases 2.0 and 3.0 while we use $thd = \{5, 4\}$ for release 2.1. This results in two groups. Each group contains three datasets, one for each release. The reason why a different set of thresholds is chosen for release 2.1 is that we would like to keep similar class distributions for the datasets in the same group. All datasets contain 209 attributes (208 independent attributes and 1 dependent attribute). Table I shows the characteristics of the datasets after transformation for each group. These datasets exhibit different distribution of class skew (i.e., the percentage of *fp* examples).

B. Results and Analysis

The results (in terms of AUC) of the correlated-based feature subset selection technique combined with random undersampling averaged over 10 runs of 5-fold CV for each dataset are reported in Table II, which contains the results for all five learners and three combination approaches. For a given learner, the best combination approach is highlighted in **bold** for each dataset. Among the 30 best performers, 23 are from Approach 1, 6 from Approach 3 and the remaining one from Approach 2.

Fig. 2 provides comparisons of three combination approaches along with various classification algorithms averaged over the respective groups of datasets. The charts intuitively demonstrate that

- Approach 1 performed better than the other two approaches for all the learners in Eclipse 1 (see Fig. 2(a)).

TABLE II
CLASSIFICATION PERFORMANCE

(a) Eclipse 1

Release	Approach	NB	MLP	KNN	SVM	LR
2.0	1	0.8437	0.8513	0.8738	0.8772	0.8657
	2	0.8234	0.8301	0.8555	0.8488	0.7626
	3	0.8205	0.8011	0.8606	0.8458	0.7193
2.1	1	0.8312	0.8612	0.8688	0.9031	0.8730
	2	0.8204	0.8327	0.8507	0.8734	0.7894
	3	0.8319	0.8371	0.8867	0.8861	0.7885
3.0	1	0.8891	0.8844	0.8834	0.9146	0.9097
	2	0.8843	0.8605	0.8747	0.9075	0.8445
	3	0.8783	0.8696	0.8628	0.9068	0.7721

(b) Eclipse 2

Release	Approach	NB	MLP	KNN	SVM	LR
2.0	1	0.8273	0.8654	0.8705	0.9064	0.8880
	2	0.8302	0.8624	0.8736	0.8807	0.8383
	3	0.8397	0.8580	0.8656	0.8934	0.7865
2.1	1	0.8219	0.8509	0.8377	0.8909	0.8818
	2	0.8150	0.8355	0.8364	0.8657	0.8395
	3	0.8119	0.8363	0.8544	0.8828	0.8548
3.0	1	0.8766	0.8963	0.8915	0.9336	0.9348
	2	0.8708	0.8951	0.8878	0.9180	0.9186
	3	0.8777	0.9025	0.9006	0.9222	0.9268

TABLE III
ANOVA FOR THE ECLIPSE DATASETS

(a) Eclipse 1

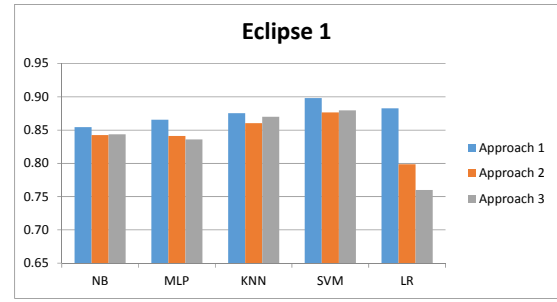
Source	Sum Sq.	d.f.	Mean Sq.	F	<i>p</i> -value
Approach	0.1217	2	0.0609	25.89	0.000
Error	1.0508	447	0.0024		
Total	1.1725	449			

(b) Eclipse 2

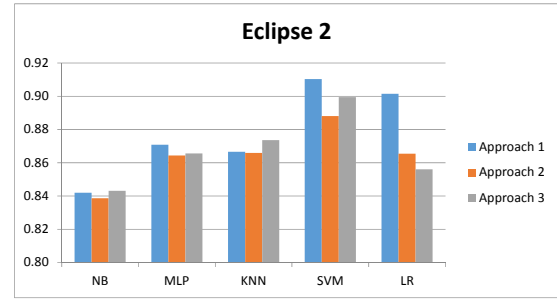
Source	Sum Sq.	d.f.	Mean Sq.	F	<i>p</i> -value
Approach	0.0156	2	0.0078	5.20	0.006
Error	0.6701	447	0.0015		
Total	0.6856	449			

- Approach 1 performed better than the other two approaches for the MLP, SVM, and LR learners in Eclipse 2, while for the NB and KNN learners, Approach 1 displayed similar or slightly worse performance than Approach 3 (see Fig. 2(b)).
- The advantage of Approach 1 is obvious when the SVM and LR learner were employed.
- Some learners, like LR, are significantly affected by the combination approach adopted, while others, like NB and KNN, are more robust with different approaches.

We further carried out a one-way analysis of variance (ANOVA) F-test on the classification performance to examine if the three combination approaches are statistically different or not. Note that all the statistical analysis was performed over each individual group of datasets, since each group displayed a distinct degree of class imbalance. In addition, as learner is not the focus of this paper, the factor taken into account only is the three combination approaches. The null hypothesis for the ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of



(a) Eclipse 1



(b) Eclipse 2

Fig. 2. Comparisons of three approaches

means is different. Table III shows the ANOVA results. The *p*-value is less than the cutoff 0.05 for the factor, meaning that the alternate hypothesis is accepted, namely, at least two approach means are significantly different from each other.

We further conducted a multiple comparison test on the factor with Tukey's honestly significant difference (HSD) criterion. For both the ANOVA and multiple comparison tests, the significance level was set to 0.05. Fig. 3 shows the multiple comparisons for both groups of datasets. The figures display graphs with each group mean represented by a symbol (o) and 95% confidence interval as a line around the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. The assumptions for constructing ANOVA and Tukey's HSD models were validated. From these figures we can see the following points:

- Approach 1 had significantly better classification performance than Approaches 2 and 3 for both groups of datasets.
- Approach 2 and Approach 3 showed similar performance (no significant difference). Approach 2 performed slightly better than Approach 3 for Eclipse 1, while Approach 2 had slightly worse performance than Approach 3 for Eclipse 2.

Overall, when the correlation-based feature selection technique is used along with the random undersampling method, we strongly recommend the data pre-processing approach in which sampling is performed prior to feature selection and the training data is formed using selected features along with unsampled data. This approach is especially effective when SVM and LR are used as classifiers.

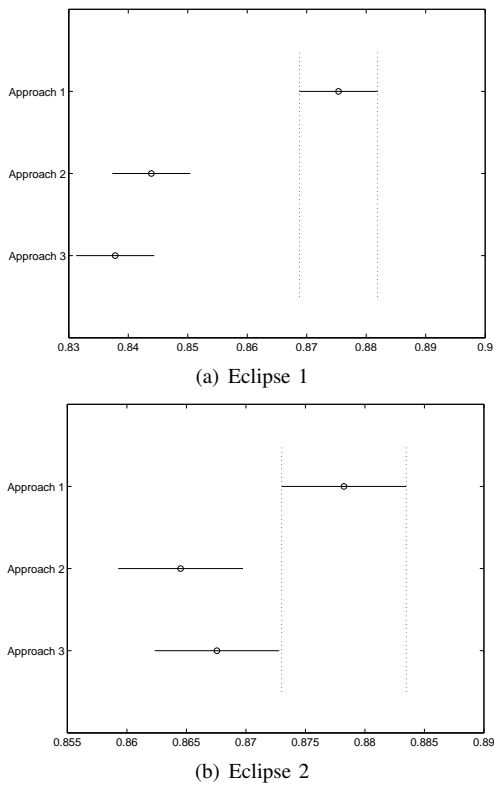


Fig. 3. Multiple comparison for three approaches

V. CONCLUSION

In this study, we proposed feature subset selection combined with data sampling to overcome the high dimensionality and class imbalance problems that often affect software quality classification. Three approaches were investigated: 1- sampling performed prior to feature selection, retaining the unsampled data instances; 2- sampling performed prior to feature selection, retaining the sampled data instances; and 3- sampling performed after feature selection. More specifically, we were interested in investigating the correlation-based feature selection method used along with random undersampling and studying the effect of three combination approaches.

In the experiments, we applied these techniques to six datasets from a real-world software system. We built classification models using five learners. The results demonstrate that among the three data pre-processing approaches, sampling performed prior to feature selection and retaining the unsampled data (Approach 1) had significantly better performance than sampling performed after feature selection (Approach 3) or sampling performed prior to feature selection but retaining the sampled data (Approach 2). Of the five learners, Support Vector Machine presented the best performance, while Multilayer Perceptron and K Nearest Neighbors demonstrated average performance. Logistic Regression performed variously with respect to different data pre-processing approaches. In contrast, Naïve Bayes showed relatively consistent performance for various approaches.

Future work will involve comparisons between feature rank-

ing and feature subset selection as well as between wrapper subset selection and filter subset selection.

REFERENCES

- [1] A. K. Pandey and N. K. Goyal, "Predicting fault-prone software module using data mining technique and fuzzy logic," *Special Issue of International Journal of Computer and Communication Technology*, vol. 2, no. 2-4, pp. 56–63, 2010.
- [2] H. Liu, H. Motoda, R. Setiono, and Z. Zhao, "Feature selection: An ever evolving frontier in data mining," in *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining*, Hyderabad, India, 2010, pp. 4–13.
- [3] M. A. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, The University of Waikato, Hamilton, New Zealand, 1999.
- [4] G. M. Weiss, "Mining with rarity: A unifying framework," *SIGKDD Explorations*, vol. 6, no. 1, pp. 7–19, 2004.
- [5] V. Kumar and S. Minz, "Feature selection: A literature review," *Smart Computing Review*, vol. 4, no. 3, pp. 211–229, June 2014.
- [6] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437 – 1447, Nov/Dec 2003.
- [7] K. Gao, T. M. Khoshgoftaar, and N. Seliya, "Predicting high-risk program modules by selecting the right software measurements," *Software Quality Journal*, vol. 20, no. 1, pp. 3–42, 2012.
- [8] C. Akalya devi, K. E. Kannammal, and B. Surendiran, "A hybrid feature selection model for software fault prediction," *International Journal on Computational Sciences and Applications*, vol. 2, no. 2, pp. 25–35, Apr. 2012.
- [9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and P. W. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.
- [10] R. Barandela, R. M. Valdovinos, J. S. Sanchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?" *In Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR'04)*, *Lecture Notes in Computer Science 3138*, no. 806-814, 2004.
- [11] R. S. Wahono, N. Suryana, and S. Ahmad, "Metaheuristic optimization based feature selection for software defect prediction," *Journal of Software*, vol. 9, no. 5, pp. 1324–1333, May 2014.
- [12] K. Gao and T. M. Khoshgoftaar, "Software defect prediction for high-dimensional and class-imbalanced data," in *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011)*, *Eden Roc Renaissance, Miami Beach, USA, July 7-9, 2011*, 2011, pp. 89–94.
- [13] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 40, no. 1, pp. 185–197, 2010.
- [14] I. H. Witten, E. Frank, and M. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.
- [15] J. Shawe-Taylor and N. Cristianini, *Support Vector Machines*, 2nd ed. Cambridge University Press, 2000.
- [16] Y. Jiang, J. Lin, B. Cukic, and T. Menzies., "Variance analysis in software fault prediction models," in *Proceedings of the 20th IEEE International Symposium on Software Reliability Engineering*, Bangalore-Mysore, India, Nov. 16-19 2009, pp. 99–108.
- [17] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the 29th International Conference on Software Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, p. 76.

A Software Defect-Proneness Prediction Framework: A new approach using genetic algorithms to generate learning schemes

Juan Murillo-Morera
Department of Informatics
National University of Costa Rica
Heredia, Costa Rica
juan.murillo.morera@una.cr

Marcelo Jenkins
Department of Computer Science
University of Costa Rica
San José, Costa Rica
marcelo.jenkins@ecci.ucr.ac.cr

Abstract

Recently, defect prediction software is an important research topic in the software engineering field. The demand for development of good quality software has seen a rapid growth in the last few years. The software measurement data collected during the software development process include valuable information about software projects status, progress, quality, performance, and evolution. The software fault prediction in the early phases of software development can help and guide software practitioners to focus the available testing resources on the weaker areas during the software development. **OBJECTIVE:** This paper presents an approach that combines three phases: data preprocessing, attribute selector and learning algorithms using a genetic approach and select the best combination. **METHOD:** The framework is comprised of 1) scheme learning generator. This component evaluates performance of the learning schemes and suggests the best option according to each data set analyzed, 2) defect predictor component builds models according to the evaluated learning schemes and predicts software defects with new data agreed to the constructed model. **CONCLUSIONS:** The framework has considered more combinations of learning schemes than other proposals which select the model configuration manually, which means that there are more possibilities to find better learning schemes for each data set. The computational processing of the genetic approach was less costly than Song approach. Finally, The Genetic approach presented an improvement of 0.032 equivalent to 3.2% more than Song approach.

Index terms— software metrics, learning schemes, genetic algorithms, fault prediction models, software quality.

doi:10.18293/SEKE2015-099

I. INTRODUCTION

Software fault prediction has been an important research topic in the software engineering field for more than 30 years [1]. The software measurement data collected during the software development process include valuable information about software projects status, progress, quality, performance, and evolution. Software fault prediction models is a significant part of software quality assurance and commonly used to detect faulty software modules, based on software measurement data (software metrics) [2], [3], [4].

Current defect prediction that works on: estimating the number of defects remaining in software systems, discovering defect associations, and classifying the defect-proneness of software components, typically into two classes, defect-prone and non defect-prone [1].

The first approach, employs statistical methods to estimate a number of defects or defect density [5], [6]. The prediction result can be used as an important measure for the software developer and can be used to control the software process, for example, decide whether to schedule further inspections or pass the software artifacts to the next development step. The second approach, borrows association rule mining algorithms from the data mining community to reveal software defect associations [7]. The third approach, works classifying software components as defect-prone and non-defect-prone, means of metric based classification: [8] and [2]. Being able to predict which components are more likely to be defect-prone supports better targeted testing resources and therefore improved efficiency. Unfortunately, classification remains a largely unsolved problem. In order to address this, researchers have been using increasingly learning schemes that include data preprocessing, attribute selector and learning algorithms. The main problem is how to select the best learning scheme, according to specific data set?. Actually does not exist a proposal that

uses genetic algorithm with the objective to select the best learning scheme configuration using a specific data set. The learning schemes have an important problem, it is how to select a correct combination of data preprocessing, attribute selection and learning algorithm for a particular data set. This novel framework has the capacity to combine different learning schemes with the objective to find the best solution according to the parameters selected and the evaluation of the performance metrics proposed.

The general objective of this research is to propose a Defect-Proneness Prediction Framework with two specific components: learning scheme generator and defect predictor.

The remainder of the article is structured as follows. Section 2 presents the background. Section 3 presents the related work. The proposed framework is presented in Section 4. Section 5 genetic approach. Section 6 experimental setup. Finally, Section 7 conclusions and future work.

II. BACKGROUND

A. Metrics

Defect predictors from static code attribute were used and defined by McCabe [9] and Halstead [10]. McCabe and Halstead are **module-based metrics**, where a module is the smallest unit of functionality (In other computational languages, modules may be called **function** or **method**). The static code attributes are useful, easy to use, and widely used.

Useful. The static code attributes have been used for the prediction of software projects with similar characteristics [1], [11]. **Easy to use.** Static code attribute like lines of code and the McCabe/Halstead attribute can be automatically and cheaply collected, even for very large systems. By contrast, other methods, such as manual code reviews, are labor-intensive. Depending on the review methods. **Widely used.** Many researchers use static attribute to guide software quality predictions: [1], [12], [13].

Halstead attribute were derived by Maurice Halstead in 1977. He argued that modules that are hard to read are more likely to be fault prone. Halstead estimates reading complexity by counting the number of operators and operands in a module. A complete reference [11].

An alternative to Halstead attributes, are the complexity attributes proposed by Thomas McCabe in 1976. Unlike, Halstead and McCabe argued that the complexity of pathways, between module symbols is more than just a count of the symbols. A complete reference [11].

B. Data sets

The most frequent data sets used by software fault prediction researchers are: CM1, JM1, KC1, KC2, KC3, KC4, MW1, MC1, MC2, PC1, PC2, PC3, PC4, PC5, AR1, AR3, AR4 and AR6. These data sets have been evaluated respect to metrics, number of attribute, number of modules among other parameters [1].

C. Learning Schemes

The Learning schemes are composed by: data preprocessing, attribute selector and learning algorithms [1]. **Data preprocessing:** It is very important to build learners. The data are pre-processed, such as removing outliers, handling missing values, and discretizing or transforming numeric attribute. **Attribute selection:** It is important when the data set may not have originally been intended for defect prediction. Not all the attributes may be helpful for defect prediction. Attribute selection methods can be categorized as filters or wrappers [15]. **Learning algorithms:** Once attribute selection has been completed, the best attribute subset are processed. Then the data set represents those attribute subset and the learning algorithm are used to build the learner.

III. RELATED WORK

Traditionally, many researchers have explored issues like the relative metrics of McCabe's cyclomatic complexity, Halstead's software science measures, and lines of code counts for building defect predictors. However, Menzies et al. [11], published a study in 2007 in which they compared the performance of two machine learning techniques (Rule Induction and Naive Bayes) to predict software components containing defects. To do this, they used the NASA MDP repository, which, at the time of their research, contained 10 separate data sets. They claimed that "such debates are irrelevant since how the attributes are used to build predictors is much more important than which particular attributes are used" and "the choice of learning method is far more important than which subset of the available data is used for learning"

Song et al. [1] published a study in which they proposed a fault prediction framework based on Menzies study but analyzing only 12 learning schemes. They argued that although "how is more important than which". The choice of which attribute subset is used for learning is not only circumscribed by the attribute subset itself and available data, but also by attribute selectors, learning algorithms, and data preprocessors. It is well known that there is an intrinsic relationship between a learning method and an attribute selection method.

Malhotra [14] published a systematic review in which she proposed as future work “There are very few studies that examine the effectiveness of evolutionary algorithms”. She points out “The future studies may focus on the predictive accuracy of evolutionary algorithms for software fault prediction”.

The aim of this paper is to build a framework that generates learning schemes using genetic algorithms. Previous works have analyzed relationships between data preprocessing, attribute selection and learning algorithms. They have used a few combinations, mainly because the evaluation of the learning schemes is processed manually, selecting the combinations. The novel proposed framework tries to select the best combination per data set, according to AUC value(maximum value).

IV. PROPOSED FRAMEWORK

It is very important before building defect prediction model(s) to decide which learning schemes should be used to construct the model. Thus, the predictive performance of learning scheme should be determined for future data. This novel framework is based on Song framework [1]. The proposed framework uses the methodology of Song except how this select the learning schemes. The novel framework consists of two components: 1) Learning Schemes Generator and 2) Defect Prediction. Figure 1, contains the details.

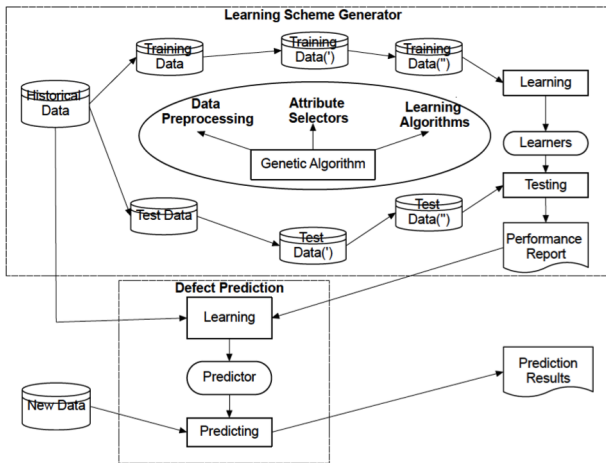


Figure 1. Fault prediction framework with genetic implementation

A. Learning Schemes Generator

The Learning Schemes Generator is a fundamental part of the software defect prediction framework. At this stage,

different learning schemes are evaluated, the best one is selected. A Genetic algorithm is used to select the best learning scheme for each data set analyzed based on their AUC. The historical data (represented by 90% of the original data) was divided into training and test data applying a MxN cross-validation based on [1].

The main steps of the Learning Scheme Generator are:

- Each data set is divided into two parts: One part is used as historical data and the other part is viewed as the new data. The historical data are represented by 90% of the original data, while the new data are represented by 10% of the original data.
- The historical data is divided into training set and test set, using a MxN cross-validation.
- The learning scheme elements (data preprocessing, attribute selector and learning algorithm) are selected by a genetic approach considering the fitness function.
- Data preprocessing is applied to both: training and test set. The test set is selected by the genetic algorithm. The result is training data(1) and test data(1) (see Figure 1).
- Attribute selector is applied to only training set and the best subset of attribute is applied to training and test set. The result is training data(2) and test data(2) see Figure 1. The attribute subset is computed interactively using a Filter strategy with NxM cross-validation different than Song, who used Wrapper evaluation. The difference is that Wrapper is computationally more costly. This a task of the genetic approach.
- Learning algorithm are build with a training set, and evaluated with a test set. This a task of the genetic approach.

B. Defect Prediction

The defect prediction is part of the proposed framework, consists of predictor construction and defect prediction. The inputs in this stage are: newData, is a data set that represents the new datas. It represents the 10% of the whole data. The other input is the HistoricalData that represents a data set with the other 90% data. Finally, the last input is the learned scheme selected by genetic algorithm. The final results are a log file with two labels: actual value and predicted value.

The main steps of the defect prediction are:

- This component uses the learning scheme selected by genetic algorithm in the previous stage.

- A predictor is build with the selected learning scheme. The whole historical data is used (not apply NxM cross-validation). All the historical data is used to build the predictor, it is expected that the constructed predictor has stronger generalization ability.
- After the predictor is build, new data are preprocessed in the same way as historical data, then the constructed predictor can be used to predict software defect with preprocessed new data.

C. Difference between the approach proposed and Song approach

The approach proposed is based on Song methodology [1]. The contributions of this novel proposal are:

- Song framework only works with 12 learning schemes. The proposed framework works with more combinations. Our maximum search space is: Data preprocessing = 7, attribute selector = 40 and learning algorithms = 41 in total ($7 \times 40 \times 41$) = 11480. Selecting automatically the best learning scheme per data set, while Song framework selects the learning scheme manually working with backward elimination and forward selection.
- Song framework works with Wrapper in the attribute selection. This computationally is very costly. The proposed framework works with Filter using NxM cross-validation.
- Song framework has a scheme evaluation component. The outcome of this component is a performance report. The proposed framework has a generator of schemes and the outcome is the best scheme learning per data set processed.

V. GENETIC SETUP

The genetic approach to be explained in the following sections: Chromosome, Operators and Fitness Function.

A. Chromosome

The chromosome is represented by a binary chain of 0s and 1s. The representation is a triple of $\langle DP, AS, LA \rangle$ that genetically is modified. The first part of the chromosome represents the data pre-processing. For the data pre-processing(DP) there are 7 possibilities, represented by a binary chain of $2^3 = 3$ bits. Additionally, for the attribute selector(AS), there are a maximum of 40 metrics (Hasteald, McCabe and LOC), representing a binary chain maximum

of $2^6 = 6$ bits. A bit with value 1 represent that this metric is present in the data set, while a bit with value 0 that it is not represented. Finally for the learning algorithms (LA) there are 41 different possibilities, representing a binary chain maximum of $2^6 = 6$ bits.

B. Operators

The operators of selection, reproduction, crossover and mutation used were applied using the following configuration: (Population size = 50, Generations = 100, Crossover probability = 0.6, Mutation probability = 0.1 and Elitism 2%)

C. Fitness Function

The Fitness was defined: $f(x) = AUC$ value. AUC value (min 0 - max 1) is defined by X-Axis and Y-Axis. It is an Area Under Curve. Y-Axis is represented by True Positive Rate or Sensitivity and X-Axis is represented by False Positive Rate or (1-Specificity).

VI. EXPERIMENTAL SETUP

A. Data sets

The data sets used were taken from the public NASA MDP repository. This study used 10 data sets: CM1, KC3, MW1, PC1, PC2, PC3, PC4, KC1, MC1 and MC2.

B. Performance Measures

This study has used the metric AUC for the comparison between approaches. Table 1 shows a complete description about this metric.

Table 1. Metrics

Name	Description	Representation
TPR	Sensitivity	$TP/(TP + FN)$
FPR	1- Specificity	$1 - (TN/(TN + FP))$
AUC	Area Under Curve	Plot Specificity (X-axis) Plot Sensitivity (Y-axis)

However, AUC was selected for the comparison to respect other approaches [1].

C. Learning Schemes

The learning schemes used in this study were:

- *Data Preprocessing (DP)*: Replace Missing Values, Math Expression, RandomSubset, Remove, Standardize, Numeric transform and Numeric to nominal.

- *Attribute selector (AS)*: This selection is a task of the genetic approach. A subset of metrics or predictors variables are selected according to the genetic selection.
- *Learning Algorithm (LA)*: The families are: Bayes (9), Functions (9), Rules (9) and Trees (14). A complete reference [15]

D. Baseline and Comparison

The Song study [1] (Song-Approach) was selected as baseline for this article. The mean of Backward Selection (BS) algorithm and the mean of Forward Elimination (FE) algorithm were calculated and the best result compared with the Genetic approach (G-Approach). Table 2 shows the comparison results with the same decimal representation [1].

Table 2. AUC-Average

Data set	Attributes	Modules	Song-Approach	G-Approach
CM1	38	344	0.634 (1)	0.728 (5)
KC3	40	200	0.689 (10)	0.679 (17)
MW1	38	759	0.635 (7)	0.697 (2)
PC1	38	264	0.711 (15)	0.727 (8)
PC2	37	1585	0.684 (11)	0.635 (14)
PC3	38	1125	0.687 (18)	0.718 (16)
PC4	38	1399	0.789 (13)	0.829 (6)
KC1	22	2096	0.697 (4)	0.778 (20)
MC1	39	9277	0.828 (19)	0.858 (12)
MC2	40	127	0.654 (5)	0.679 (9)

Table 2 shows that G-Approach had a better performance according to AUC metric. The G-Approach presented an average of 0.732 while Song-Approach presented an average of 0.700. The difference between G-approach and Song-approach has been an improvement of 0.032 equivalent to 3.2%. The order of the runs are represented by tiny numbers (see Table 2).

E. Statistical Analysis and Discussion

Figure 2 shows that Genetic approach is better than Song approach per each data set except PC2 and KC3, where the model proposed presented less performance. The MC1 data set has presented the best performance with AUC = 0.85. Additionally, other datasets where the Genetic approach presented better performance were: MW1, PC1, PC3, PC4, KC1, MC1 and MC2.

The data sets with the worst performance in the Genetic approach were: PC2 with a difference of 0.045 respect to Song approach and KC3 with a difference of 0.01 respect to Song. The data set that presented with the Genetic approach

the best performance was CM1 with a difference of 0.094. (see Figure 2).

The order of runs was random. Twenty runs was executed using the model proposed in section IV.

The first factor used was the framework. This factor has been represented by two levels (Genetic and Song). Otherwise, the second factor was the data set and has been represented by ten levels: CM1, KC3, MW1, PC1, PC2, PC3, PC4, KC1, MC1 and MC2.

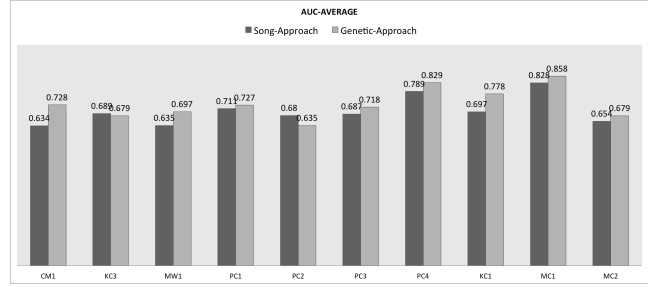


Figure 2. Performance Frameworks

The hypotheses were:

- Hypothesis 1: test the relationship between frameworks according to their performance
 H_{0frm} : Are there significant difference between frameworks respect to AUC?
 H_{1frm} : Are there not significant difference between frameworks respect to AUC?
- Hypothesis 2: test the relationship between data sets according to their performance
 H_{0ds} : Are there significant difference between data sets respect to AUC?
 H_{1ds} : Are there not significant difference between data sets respect to AUC?

Wilcoxon signed rank test was applied for the first hypothesis. A significant difference was found in H_{0frm} . This means that H_{0frm} is rejected and exist a difference statistically significant. The $pvalue$ reported was $pvalue = 0.04883 < \alpha = 0.05$. This represented that Genetic approach was better than Song approach, 0.7328 and 0.7004 respectively. Further, the Genetic approach was computationally less costly than Song approach, because the genetic approach has implemented the evaluation with the filter strategy while Song approach has implemented the evaluation with the wrapper strategy.

The second hypothesis was evaluated with an one-way anova study. The first step of this study has been the study of normality, homogeneity of variances and independence assumption. Shapiro-Wilk and Bartlett test were

applied. The results for both test were: normality test, $p_{value} = 0.9051 > \alpha = 0.05$ and homogeneity of variances test $p_{value} = 0.898 > \alpha = 0.05$ (framework) and $p_{value} = 0.90 > \alpha = 0.05$ (data set). The independence principle was assumed. This means not violation of normality assumption. Then the next step was the validation of p_{value} for H_{ds} . A significant difference was found into H_{ds} , this reported a $p_{value} = 0.0496 < \alpha = 0.05$. This means that H_{0ds} is rejected and a Fisher Test can be applied. The Fisher test presented the following results:

Table 3. Group of data sets

Group	DataSets
Group 1	MC1
Group 2	PC4
Group 3	PC1, KC1
Group 4	CM1, PC3
Group 5	KC3, MW1, PC2, MC2

Table 3 shows the groups with significant difference. All the data sets from different groups have presented significant difference. The group with more data sets is the group-5, and the rest of the groups have been represented with one or two data sets. An important issue is the interval representation. For example, Genetic approach presented an AUC value: min = 0.635, max = 0.858 while Song approach presented an AUC values: min = 0.635, max = 0.828.

VII. CONCLUSIONS AND FUTURE WORK

The framework has included more combinations of learning schemes than other proposals. This means, there are more possibilities to find better learning schemes for each data set. The genetic approach has presented better performance than Song approach in the majority of the cases, representing eight of ten data sets. The data set where the Genetic approach had less performance was PC2 while the data set with more performance was MC1.

The predominant learning schemes were: DP= {Replace Missing Values, Math Expression}, AS={Genetic selection} and LA={Naive Bayes, Decision Tree, Lineal Regression, Boosting, Bagging, Support Vector Machine}.

Another important conclusion has been the size of the data sets. For example MC1 is the data set with more size (9277 modules). This data set represented the best AUC (0.858) in the Genetic approach. A similar situation with KC1, this data set represented the second in size (2096 modules) and the second with the best AUC (0.82). This situation is different in Song approach where the AUC was reported with a value of 0.82 (MC1) and (0.69) KC1 respectively.

As Future work, it is necessary to include more data sets with different size, noise level and imbalance data from

public and private repositories. It is very important more experimentation with different parameters configuration and others methods of crossover and mutation that improvement the performance.

VIII. ACKNOWLEDGMENTS

This research was supported by Doctoral Program in Computer Science at the University of Costa Rica. Costa Rican Ministry of Science, Technology and Telecommunications (MICITT).

REFERENCES

- [1] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 356–370, 2011.
- [2] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "Software measurement data reduction using ensemble techniques," *Neurocomputing*, vol. 92, pp. 124–132, 2012.
- [3] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *Software Engineering, IEEE Transactions on*, vol. 38, no. 6, pp. 1276–1304, Nov 2012.
- [4] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.
- [5] J. Munson and T. Khoshgoftaar, "Regression modelling of software quality: empirical investigation," *Information and Software Technology*, vol. 32, no. 2, pp. 106–114, 1990.
- [6] N. B. Ebrahimi, "On the statistical analysis of the number of errors remaining in a software design document after inspection," *Software Engineering, IEEE Transactions on*, vol. 23, no. 8, pp. 529–532, 1997.
- [7] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," *Software Engineering, IEEE Transactions on*, vol. 32, no. 2, pp. 69–82, Feb 2006.
- [8] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," *Applied Soft Computing*, vol. 21, pp. 286–297, 2014.
- [9] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, December 1976.
- [10] M. Halstead, *Elements of Software Science*. Elsevier, 1977.
- [11] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 2–13, Jan 2007.
- [12] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 432–441.
- [13] R. Malhotra and A. Jain, "Fault prediction using statistical and machine learning methods for improving software quality," *JIPS*, vol. 8, no. 2, pp. 241–262, 2012.
- [14] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.
- [15] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

Using Time Series Models for Defect Prediction in Software Release Planning

James Tunnell and John Anvik
Computer Science Department
Central Washington University
Ellensburg, WA 98926, USA
[tunnellj, janvik]@cwu.edu

Abstract—A time series model is presented that uses historical project information to predict the number of future defects, given the number of proposed features and improvements to be completed. This allows for hypothetical release plans to be compared by assessing their predicted impact on testing and defect-fixing time. We selected the VARX time series model as a reasonable approach. The accuracy of the model appeared low for a single dataset, but the error was found to be normally distributed.

Keywords—*software defect prediction; quality assurance; release planning; time series model;*

I. INTRODUCTION

There are two primary concerns in software release planning: improving functionality and maintaining high quality. Both objectives are constrained by limits on development time and budget, so the scope of the planned work must be limited to accommodate fixing inevitable defects (bugs) that will arise. In this way, a high quality software product can be produced while also improving its functionality.

A significant consideration in the release planning process is the amount of time allocated for testing and bug-fixing. If this factor is not considered, the project risks a slip in the schedule or in the quality of the product. As the time and effort required for testing and bug-fixing will likely be a function of the defects introduced during development, it is desirable to be able to predict the number of expected defects.

A potential application for defect prediction is to compare different release plans according to their estimated bug fallout and subsequent impact on testing and bug-fixing times. This would assist release planners in ensuring that the total development time does not exceed the project's time budget for a release. The comparison of different release plans is integral to release plan optimization, which is the focus of The Next Release Problem [2], a key problem in Search-Based Software Engineering (SBSE) [9, 13].

Most approaches to defect prediction focus on either code analysis [1, 5, 6, 8, 11] or historical defect information [7, 10, 13]. However, for the defect prediction model to be useful in comparing release plans, the model should also depend on the planned features and improvements planned for the next release, as well as the defects from past releases.

This paper presents an approach to defect prediction that can be applied for a proposed release. A multivariate time series model is used that incorporates information about proposed features, improvements, and historical defect data.

The paper proceeds as follows. First, Section II presents further motivation for the use of a time series model for predicting defects. Next, we present an overview of concepts in time series modeling in Section III. Section IV presents our modeling methodology and Section V presents the application of the approach, which is applied to a software project dataset. Related work is presented in Section VI, and the paper concludes in Section VII.

II. MOTIVATION

Release planners typically rely on both their experience and project conventions to generate a release plan by selecting planned features and improvements such that the estimated time to test for and fix defects will not cause a schedule slip.

However, if the defect estimation technique is only loosely based on past experience, as with a rule-of-thumb, then it may prove too coarse for comparing multiple release plans, and may not provide any quantitative difference between release plans that are similar (but not the same). Even for dissimilar release plans, such an approach still has the disadvantage of lacking confidence intervals to quantify prediction uncertainty.

An alternative approach is to develop a model that will take into account the differences in composition of features and improvements between the release plans. Such a model would assume some explanatory relationship.

Since predictive models rarely have perfect accuracy, confidence levels are an important part of any prediction to allow release planners to assess the risk of relying on the defect prediction. Planners can choose a more narrow prediction window, in exchange for a larger risk that the prediction is inaccurate. Conversely, a wider prediction window means that the potential cost range is also wider with a lower risk of inaccuracy.

III. TIME SERIES MODELING

In this section, time series and autoregressive models are introduced. Then, further concepts related to modeling, exogeneity and stationarity, are discussed.

A. Time Series

A time series is a collection of observations that occur in order, with an underlying process that is stochastic. Critically, the sequence of observations cannot be re-arranged, as each observation is typically dependent on one or more previous observations. This dependence is termed autocorrelation and accounting for it is one of the primary functions of a time series model.

B. Autoregressive Models

A basic autoregressive (AR) model is formed as a linear combination of previous values, plus a white noise term that accounts for random variations (the stochastic portion). When the AR model is extended to the multivariate case (i.e. allowing for multiple time series), a Vector AR (VAR) model is formed. This model will support not only a time series for defect count, but also time series for the two release plan variables: improvements and new features.

The VAR model can be further extended by considering one or more variables to be exogenous, making a VARX model. Exogenous variables are used to explain the other non-exogenous variables, but the model does not attempt to explain the exogenous variables themselves. This model meets the requirements of the explanatory model described in the Motivation section, since it would allow release plan variables to be kept exogenous and used only to explain defect count.

C. Stationarity and Trends

A strictly stationary process has a probability distribution that is time-invariant. This means statistics such as mean and variance do not change. The AR, VAR, and VARX models discussed so far require time series data that is stationary, where the probability distribution of the underlying stochastic process is time-invariant. Testing can identify a time series as being stationary, trend stationary, or non-stationary.

A time series can be established as non-stationary by testing for the presence of a unit root in the underlying AR model. The unit root test used is the Augmented Dickey Fuller (ADF) test. On the other hand, a stationarity test establishes a time series as trend stationary by testing for the presence of a deterministic trend function (either a constant or a line). The stationarity test used is the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test.

IV. MODELING METHODOLOGY

The typical methodology used for building time series models involves specification, estimation, and diagnostics checking [4, p. 478]. Once specified and estimated, the diagnostic checking step ensures that only valid models are considered for selection. The final step of modeling is selection, where the models are compared by some model selection criterion [4, p. 581]. This section presents our approach to specifying, estimating, diagnostics checking, and model selection for defect prediction.

A. Model Specification & Estimation

The specification of a VARX(p) model is accomplished by choosing an order p , which is the number of autoregressive

terms to include in the model. Then the model parameters can be estimated by a procedure such as least squares regression.

The model order will directly affect the number of parameters included in the model. One goal of specification is to avoid having too many parameters relative to the number of observations. To this end, we establish a ratio K of the number of observations to the number of parameters. By choosing a minimum value for this ratio, K_{min} , and using the formula for the number of parameters in a VARX(p) model, the following equation can be used to obtain a maximum model order p_{max} :

$$p_{max} = \left\lfloor \frac{n}{mK_{min}} \right\rfloor$$

where there are m time series variables and n samples. This establishes an upper bound on model order, so model specification will include the generation of models having order 1, 2, ..., p_{max} . These models, with their estimated parameters, will be candidates for final model selection after undergoing diagnostic checking.

B. Diagnostics Checking

Diagnostic checking is performed to verify that a model can be accepted. This step includes testing for stability and for model inadequacy. A stability test checks that the roots of the AR process characteristic equation lie outside the unit circle [4, p. 56]. To test inadequacy, the Ljung-Box is used to compare the model residuals to white noise.

C. Model Selection

Model selection criteria are used to compare models by their fit, to minimize residual error, and to penalize the model to some degree based on the number of parameters. Of the commonly used selection criteria, the standard Akaike Information Criterion (AIC) was used because “[t]he penalty for introducing unnecessary parameters is more severe for BIC and AICC than for AIC” [3]. A less severe penalty for the number of parameters would be preferred in this case, since we are already limiting the number of parameters in the model specification step, and because additional parameters may in fact be necessary to account for time series autocorrelations with higher lags.

V. APPLICATION OF METHODOLOGY

To validate our approach of using a time series model to predict defects, we used historical data taken from a software project’s issue tracking system. Issue tracking systems are used by projects for tracking development tasks, features, enhancements, and bugs, both past and present.

We chose the MongoDB¹ Core Server project as the data set. This project was chosen as it has been active since May 2009 and uses JIRA² for issue tracking, which made it easy to collect data. Issues for versions 0.9.3 through 3.0.0-rc6 were exported from the project’s JIRA web interface into XML

¹ MongoDB is an open source, document-oriented database system.

² JIRA is an issue tracking and project management system made by Atlassian.

format. The fields collected from each issue report were: type, priority, creation date, and resolution date.

Only issues marked as *fixed*, *complete*, or *done* were used for modeling. In the data collected, 18 (0.26%) issues did not meet this criterion and were excluded. Also, *JIRA* supports issues having sub-tasks. Any sub-task whose parent issue was not in the dataset was considered orphans and discarded. There were 20 (0.28%) orphaned sub-tasks in the dataset. The final dataset contained 7042 issues.

A. Data Preparation

After creation, the dataset was operated on to prepare it for time series modeling. The data was sampled, made stationary, and windowed. These three steps are discussed next.

1) Sampling

First, the data was sampled at regular periods to measure the following: number of improvements resolved, number of features resolved, and number of bugs created. A 7-day sampling period was used.

2) Establishing Stationarity

To establish stationarity, the ADF unit root and KPSS stationarity tests were applied. In both tests, it was assumed that the deterministic component was constant (without slope). These test results did not agree, so the time series data was differenced and the tests were rerun. The test results then agreed, establishing the stationarity of the differenced data.

3) Time Windowing

It can be assumed that the software development process underlying a given project changes over time. Rather than developing a model that also changes over time, the data was kept for modeling only if it occurred within a time window. This was done to limit the effect of process change on the model. A time window of 78 weeks (approximately 18 months) was selected to balance between more observations (to capture consistent long-term behaviors), and fewer observations (to limit exposure to behavioral changes).

Applying this time window, the data was divided into three 78-week windows. As the data was differenced, the first sample was skipped in each data period. These windowed periods are denoted W_{2-79} , W_{80-157} , and $W_{158-235}$.

B. VARX Modeling

1) Use of the VARX Model

The VARX model was chosen to model the time series because there are multiple time series to be considered jointly. The $Y_{\Delta imp}$ and $Y_{\Delta new}$ time series were both considered exogenous, so that hypothetical future values could be considered when comparing release plans. And by selecting $K_{min} = 4$, a maximum model order of $p_{max} = 6$ is obtained, so only model orders 1 through 6 were estimated for later diagnostic checking.

C. Model Diagnostic Checking

Candidate models were tested for stability and inadequacy. A 5% significance level was used in the Ljung-Box test. All model orders were found stable for all windowed periods.

Several model orders were found to be inadequate, specifically orders 1-2 for period W_{2-79} , and order 5 for period $W_{158-235}$.

D. Model Selection

Models that were not rejected for instability or inadequacy were then compared and the best for each windowed period was selected by AIC selection criterion. The best model orders found were 4, 1, and 1, for windowed periods W_{2-79} , W_{80-157} , and $W_{158-235}$, respectively. The fit for each of these models was demonstrated by plotting one-step predictions along with actual values, as shown for each model in Fig. 1. The fit for each appears to track well with many of the significant changes in the time series.

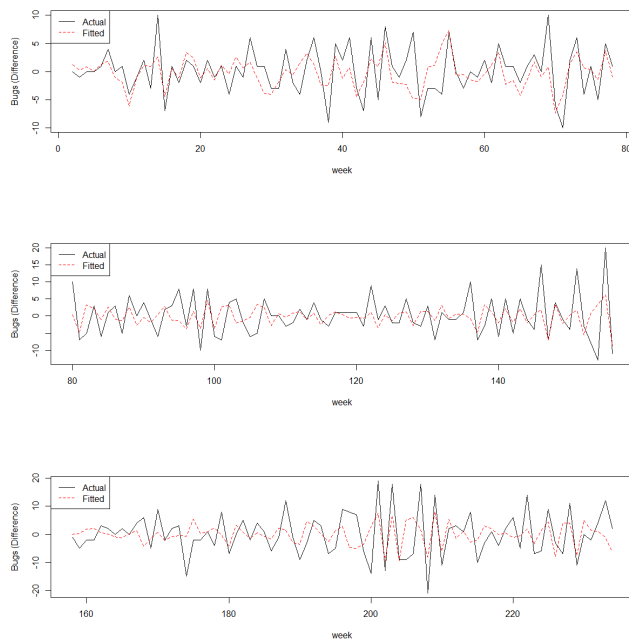


Figure 1. One-step predictions vs actual values, for each model selected by AIC score.

E. Forecasting

Selected models were used to forecast the number of defects in the next sample after the end of the window. The input for making these predictions was the number of improvements and features that were expected to be resolved.

Table I shows the resulting single-step, out-of-sample defect prediction data for the first time window, W_{2-79} , including the upper and lower bounds of the confidence intervals. The actual number of improvements, features, and bugs in the prediction sample period was 4, 0, and 18, respectively. Notice that the actual number of bugs, 18, is outside of the 90% confidence interval, which spans from 6.4 to 13.79 (see the outlined row in Table I). On the other hand, the actual number of future defects in the next window, W_{80-157} , was 17. This was inside the 90% confidence interval, which spans from 13.38 to 18.00.

TABLE I. FORECASTING AT THE END OF THE FIRST TIME WINDOW, $W_{2,79}$. FUTURE OUTPUT VALUES ARE PREDICTED FOR A NUMBER OF HYPOTHETICAL INPUT VALUES.

Improvements	Features	90% lo	75% lo	mean	75% hi	90% hi
2	0	5.61	6.72	9.31	11.89	13.00
2	1	5.54	6.66	9.24	11.82	12.93
2	2	5.48	6.59	9.17	11.75	12.86
2	3	5.41	6.52	9.10	11.69	12.80
4	0	6.40	7.51	10.09	12.68	13.79
4	1	6.33	7.44	10.03	12.61	13.72
4	2	6.27	7.38	9.96	12.54	13.65
4	3	6.20	7.31	9.89	12.48	13.59

To gauge how well prediction will work in general, a sliding 78-week window was applied, starting at the first sample period, and shifting by one sample period after modeling. Only actual numbers were used in this forecasting. The resulting distribution of errors between the mean forecasted bugs and the actual number of bugs is shown as a histogram in Fig. 2. Note that the histogram appears to be normally distributed. The actual number of bugs was inside the 90% confidence interval for 23.87% of the sliding window ranges.

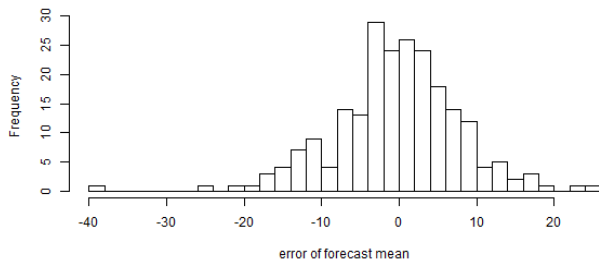


Figure 2. Histogram of forecast mean errors obtained using a 78-week sliding window.

VI. RELATED WORK

Prior defect prediction techniques generally fall into two categories: those based on code analysis and those based on statistical analysis.

Code analysis techniques typically involve a detailed analysis of code, using metrics such as lines of code (LOC) [1] or decision points [5]. Henry and Kafura [8] defined metrics from design document information for use in defect prediction.

Statistical analysis techniques create mathematical models based on historical defect occurrence information, such as regression analysis and extrapolation [10]. Graves et al. [7] developed a weighted time-damping model using a statistical analysis of change management data. And Singh et al. [12] applied the Box-Jenkins method to time series datasets from the Eclipse and Mozilla projects to predict defect counts using an ARIMA model, though their model is non-explanatory and is only in terms of past defects. We included past features and improvements as model inputs, so defects can be predicted using values for any given hypothetical release plan.

VII. CONCLUSIONS AND FUTURE WORK

The VARX modeling methodology was successfully applied to the time series data collected from the *MongoDB* project. A model was created for each of three time windows and then used to make defect predictions for a range of hypothetical values for the number of improvements and features. Also, a picture of the prediction performance was obtained by applying the approach with a sliding window. This resulted in a normally distributed error between the mean forecasted and actual number of bugs. A low proportion (23.87%) of the sliding window ranges included the actual number of bugs using a 90% confidence interval. These results indicate that the VARX model had a low prediction accuracy for the actual number of defects in the *MongoDB* dataset.

Having applied the VARX time series model to one project dataset, a next step is to apply the methodology to other software project data sets, such as *Eclipse* or *Firefox*, to better determine the applicability of the modeling approach.

REFERENCES

- [1] F. Akiyama. An example of software system debugging. In IFIP Congress (1), volume 71, pages 353–359, 1971.
- [2] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whitley. The next release problem. *Information and software technology*, 43(14):883–890, 2001.
- [3] S. Bisgaard and M. Kulahci. *Time series analysis and forecasting by example*. John Wiley & Sons, 2011.
- [4] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis*. John Wiley, 2008.
- [5] J. E. Gaffney. Estimating the number of faults in code. *Software Engineering, IEEE Transactions on*, SE-10(4):459–464, July 1984.
- [6] E. Giger, M. Pinzger, and H. C. Gall. Comparing fine-grained source code changes and code churn for bug prediction. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 83–92. ACM, 2011.
- [7] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *Software Engineering, IEEE Transactions on*, 26(7):653–661, 2000.
- [8] S. Henry and D. Kafura. The evaluation of software systems' structure using quantitative software metrics. *Software: Practice and Experience*, 14(6):561–573, 1984.
- [9] H. Jiang, J. Zhang, J. Xuan, Z. Ren, and Y. Hu. A hybrid ACO algorithm for the next release problem. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*, pages 166–171. IEEE, 2010.
- [10] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam. Empirical evaluation of defect projection models for widely-deployed production software systems. *SIGSOFT Softw. Eng. Notes*, 29(6):263–272, Oct. 2004.
- [11] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 284–292. IEEE, 2005.
- [12] L. L. Singh, A. M. Abbas, F. Ahmad, and S. Ramaswamy. Predicting software bugs using arima model. In *Proceedings of the 48th Annual Southeast Regional Conference*, page 27. ACM, 2010.
- [13] J. Xuan, H. Jiang, Z. Ren, and Z. Luo. Solving the large scale next release problem with a backbone-based multilevel algorithm. *Software Engineering, IEEE Transactions on*, 38(5):1195–1212, 2012.

An Ontology for Describing Security Events

Hossein Fani
University of New Brunswick
Fredericton, NB, Canada
hosseinfani@gmail.com

Ebrahim Bagheri
Ryerson University
Toronto, ON, Canada
bagheri@ryerson.ca

Abstract— Mining security events helps with better precautionary planning for community safety. However, incident records are expressed in diverse and application dependent formats which impedes common comprehension for automatic knowledge extraction and reasoning. In this paper, we present Security Incident Ontology, SIO, a novel light-weight domain ontology for security incidents. We use Timeline to annotate the temporal facts of incidents and adopt Event to represent any security issues from indecent behavior to assault to more adverse crime which raise the security alarm in a community. It will present a unique way to the security incident detectors, a police officer, Robocops, or intelligent CCTV cameras, to report security events. We use SIO in populating security incident notifications of Integrated Risk Management (IRM) at Ryerson University to evaluate its competency, for Ryerson University campus has both business and housing area in the vicinity and encompass not only high rate, but also wide variety of different security issues. SIO is developed in OWL 2 with Protégé.

Ontology; Semantic Web; OWL; Security Incident; Event.

I. INTRODUCTION

Environmental health and safety is the utmost priority of any municipal governor. In order to enhance the continual security of a community nearly real-time software systems are developed to monitor the area, record the events and send alarm notifications [1] [2]. For an ounce of prevention is worth a pound of cure, lots of efforts have been put to find security incidents' root cause, prediction, and prevention by mining huge datasets of records [3]. Unfortunately, these records do not comply with a widely accepted standard in representation which impede the automatic knowledge extraction and reasoning. Local and official security guards or crime detector CCTV cameras [4] express same event with different schemes. In such a situation, we desperately need an ontology which, to our knowledge, thus far, we do not have any. In this paper, we devised a novel light-weight domain ontology, Security Incident Ontology (SIO), which is able to describe any kind of security risk from indecent behavior to assault to more adverse crime. Temporal aspects of security incidents are indispensable and SIO links to Timeline [5] ontology. Timeline is an extension to OWL-Time [6] [7] [8]. Timeline together with Event [9] ontology are able to address any general events. However, in SIO we specify the Event to security incident types and Agent class to Victim and Subject. SIO is developed in OWL2 with Protégé ontology editor. To show SIO capability in answering any competency question in security paradigm, we automatically extracted security incidents from security notifications of Integrated Risk Management (IRM) system at Ryerson University during year 2014 and represent them in SIO. Ryerson University campus has both business and

housing area in its neighborhood and located at Toronto downtown. As a result, it can be a well suited area for not only high security threat rate, but also embodies wide variety of different crimes. According to our populated dataset, on a monthly basis, two security incidents of different types occur in the campus. Moreover, we are going to publish our populated dataset in Linked Data [10] to not only get it into the Linked Open Data Cloud [11], but also provoke SIO as the widely accepted representation for security events.

The remainder of this paper is organized as follows. Section 2 introduces the background and the initiatives for the work. In section 3, we construct event oriented security incident ontology. Section 4 presents the ontology evaluation by automatically populating security incident instances from Ryerson IRM notifications. Finally, the conclusions are given in section 5.

II. BACKGROUND

Ryerson University believes an informed community is a safer one. The Integrated Risk Management (IRM) system notifies all Ryerson staff, students, faculty and alumni (who have graduated within the past five years) by security incident alarms which are delivered directly via email [12]. For the urban campus is located at the downtown center of Toronto, the most populous, yet commercial capital city in Canada [13], such system seems indispensable to continually enhance the safety and security of the community. Each notification includes temporal facts of the incident, location, victim and suspect details, and a brief account of whole event. Likewise, Toronto Police Service (TPS) provides several mailing lists for which citizens of different divisions can sign up to be kept up-to-date on current happenings across the city, and in their community [14]. However, lack of standard way between reporting parties in representing security happenings and verification mechanism impedes automatic, yet reliable crime analysis and knowledge discovery for long-term planning. As a result, TPS has a disclaimer in its crime statistics webpage [15] and states firmly they make no warranty to the content, accuracy, timeliness or completeness of the statistics.

In order to provide a reliable, yet explicit understanding of incidents reported from vast variety of detectors we took an ontological approach. The ontology-based description approach is not novel. There are several initiatives to model event-focused concepts in knowledge representation [16], medical informatics [17] [18], sports [19], business news [20], scholarly events [21], life events [22] and multimedia community [23] [24]. Crime Emergency Event Model (CE2M) [25], despite what its name implies, is an ontology for emergency events

This work is funded by the Natural Sciences and Engineering Research Council of Canada.

DOI reference number: 10.18293/SEKE2015-101

rather than crimes. [26] constructs an event ontology to describe cyber-crimes on the level of event for crimes in Web such as online fraud, internet pornography, illegal trade, false advertising, violations of privacy, etc. And it is still not a general crime domain ontology. [27] describes a knowledge base of Politically Motivated Violent Events (PMVE) consisting of a domain ontology and of instance data. Although the work explain almost nothing about its ontology and focus more on extracting violent crimes from online news reports, it provides us, along with the aforementioned ontology models, an insight into how to model our generic security event structure. We reuse a most cited event ontology, Event [9], instead of recreating a new one and specifically adopt it to Security Incident Ontology, SIO, a novel explicit specification of security incident conceptualization.

III. SECURITY INCIDENT ONTOLOGY

A. Ontology Engineering

Ontology development is not an easy task. It requires skills and is still an art rather than technology. People need a sophisticated methodology to help them develop an ontology [28]. We use UPON, a methodology for ontology building derived from the Unified Software Development Process [29]. UPON is use-case driven and iterative process methodology, well-suited for developing domain ontologies. The iteration counts for inception, elaboration, construction, and transition phase are 1, 2, 2, and 1 respectively in the first release version of our ontology. Our ontology editor is Protégé. The environment not only facilitates reusing other ontologies by importing ontologies as well as its ontology library [30], but also has visualization tools which help an engineer to have big clear picture of ontology compartments. Finally, our ontology language is OWL2.

B. Event Ontology Adoption

Security incident is an event. That simply means:

```
<owl:Class rdf:ID="SecurityIncident">
  <rdfs:subClassOf rdf:resource="#Event" />
</owl:Class>
```

We either can create a new upper level core ontology for event or reuse the most cited, yet suitable one for our work. We prefer the latter one and adopt Event [9]. This ontology is based on the view expressed by James F. Allen and George Ferguson in [31] which states that events are primarily linguistic or cognitive in nature and the world contains no events. Events are just certain useful and relevant patterns of world changes. Nonetheless, this ontology has already been proven useful in a wide range of context, due to its simplicity and usability. The ontology has `event:Event` at the heart and reuses Timeline [5] ontology for temporal predicate `event:time` and Geo RDF vocabulary [32] for spatial predicate `event:place`. Timeline ontology itself has OWL-Time [6] at heart to express instantaneous or extended time object along with a temporal algebra. Additionally, it is able to describe timelines other than the universal one which may be used in a recorded track or on any media with a temporal extent. `event:factor` is everything used in an event, `event:product` is whatever produced by an event and `event:sub_event` provides a way to split a complex event into simpler ones. Fig. 1.

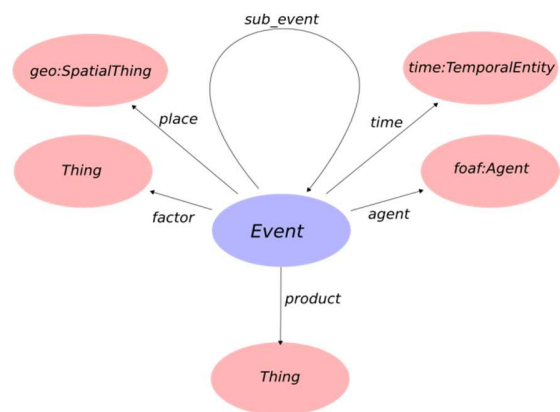


Figure 1. The Event Model. This ontology deals with the notion of reified events. It defines one main Event concept. An event may have a location, a time, active agents, factors and products.

However, security incidents have some discriminatory characteristics. The event:agents in Event ontology is a victim or suspect with this respect. event:product in security event is presumed as a crime or safety threat. Furthermore, an incident in this area may have spatial trajectory which demands change in event:place property. Thus, Event ontology should be customized to meet these needs.

C. Reification

From the survey and analysis of various security incidents, glossary of terms is formed. TABLE 1 are lists of new concepts included in our SIO and Fig. 2 shows its inter-relations. SIO leverage `event:agent` predicate of `event:Event` class for modeling the `sio:Victim` and `sio:Subject` of `sio:SecurityIncident`. We assume that security incident should have at least one `event:agent` of type `sio:Subject` which is `owl:subClassOf foaf:Agent` indirectly. This entity is the most enriched and has predicates to describe and track the subject in an incident such as. `sio:height`, `sio:weight`, `sio:hairColor`, `sio:image`, `sio:preState`, `sio:postState`. `sio:preState` explain the subject how he/she start to make a security concern and `sio:postState` shows the his/her final destiny e.g. flee, arrest, killed. `sio:Victim`, likewise, has these predicates. It is worth mentioning that on the one hand `sio:Subject` or `sio:Victim` may be `sio:CommunityMember`, and on the other hand there is no is-a or kind-of relationship between them. SIO leave it to the time when we instantiate a security incident. By then, we can add `rdf:type` to manage the security incident types. Moreover, a taxonomy of incidents has been defined. Each type of incidents has its own class.

In addition to the mentioned concepts, we found that a same incident can be reported by different report party, sometimes with time lag between them. Hence, we add similarity relation between incident entities to identify duplicated of a same incident. We eschewed the obfuscation of relativity of simultaneity [33] for the first version SIO and identify two incidents which are simultaneous in location of space (`event:place`) and location of time identical. As different source of detection fortify the incident integrity, we do not remove the duplicates and instead make an association between them by built-in OWL property `owl:sameAs`.

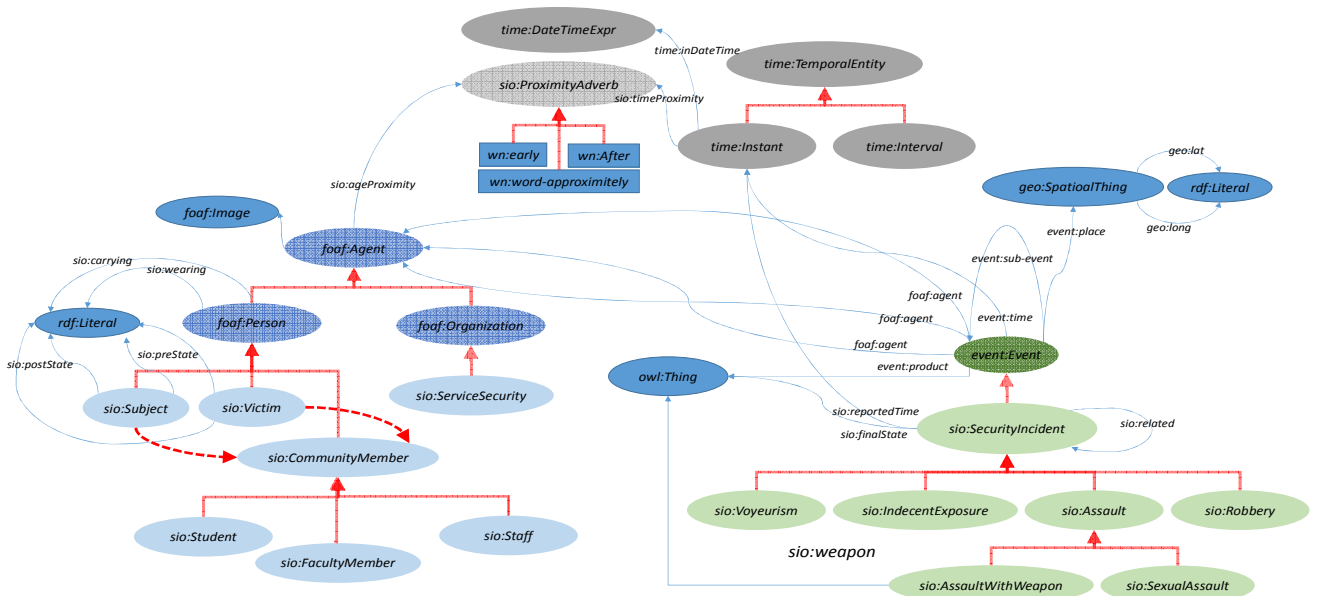


Figure 2. A brief snapshot concepts reuse from foaf, geo, wn, time, event, and our sio. Red arrows imply inheritance or owl:subClassOf and the blues are associations.

The SIO ontology specification and its Turtle version are available at <http://semionet.mnet.ryerson.ca/ontologies/sio.owl>

TABLE I. A list of the major concepts and entities (second level) in the SIO ontology version 1.0

Term	Description
Security Incident	The set of connected events which reflects an occurrence, unusual problem, incident, deviation from standard practice, or situation that requires follow-up action.
Sexual Assault	A form of sexual violence, is any involuntary sexual act in which a person is threatened, coerced, or forced to engage against their will, or any non-consensual sexual touching of a person
Assault with Weapon	In committing an assault, the subject carries, uses or threatens to use a weapon or an imitation thereof
Assault	The direct or indirect application of force to another person, or the attempt or threat to apply force to another person, without that person's permission
Indecent Exposure	Indecent exposure is the deliberate exposure in public or in view of the general public by a person of a portion or portions of his or her body, in circumstances where the exposure is contrary to local moral or other standards of appropriate behavior
Voyeurism	Voyeurism is the sexual interest in or practice of spying on people engaged in intimate behaviors, such as undressing, sexual activity, or other actions usually considered to be of a private nature
Robbery	The act of taking or another person's property (including attempts)
Victim	A victim of a security incident is an identifiable person who has been harmed individually and directly by the suspect, rather than by society as a whole
Subject	A subject is a known person who initiate the security incident usually by committing crime.

IV. EVALUATION

We believe that a successful domain ontology should I) be capable of expressing any instance data and answer any competency questions in the associated domain, II) become popular and be reused by knowledge engineers in similar domains. To show the first one, we populate the security incidents which happens at Ryerson University urban campus. Integrated Risk Management (IRM) system at Ryerson notifies its community members including staff, students, faculty, and alumni (who have graduated within the past five years) about security incidents in a near real-time manner. The notification is delivered to the email which has a hyper link to the incident specific webpage. Ryerson could have provided us with the whole incidents as this work is done partly under Ryerson affiliation. Then we could easily extract information and transform it to SIO way of representation by an Extract-Transform-Load (ETL) engine or a mapper.

However, other engineers who wants to reuse our ontology may not have same chance of data accessibility. Hence, we continue to obtain the incidents information independently by crawling the incidents webpages. This way, we accompany our SIO with its crawlers as a fully-fledged solution. Moreover, we have provided SPARQL and JDBC endpoint to our dataset in jdbc:virtuoso://semionet.mnet.ryerson.ca:1111/charset=UTF-8/log_enable=2 and <http://semionet.mnet.ryerson.ca:8890/sparql> with Graph IRI <http://ls3.mnet.ryerson.ca/SecurityIncident/test>. That means we have satisfy the 5 steps of Linked Open Data. The steps (stars) are [34]:

- ★ Available on the web (whatever format) but with an open license, to be Open Data
- ★★ Available as machine-readable structured data (e.g. excel instead of image scan of a table)
- ★★★ non-proprietary format (e.g. CSV instead of excel)
- ★★★★ Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff
- ★★★★★ Link your data to other people's data to provide context

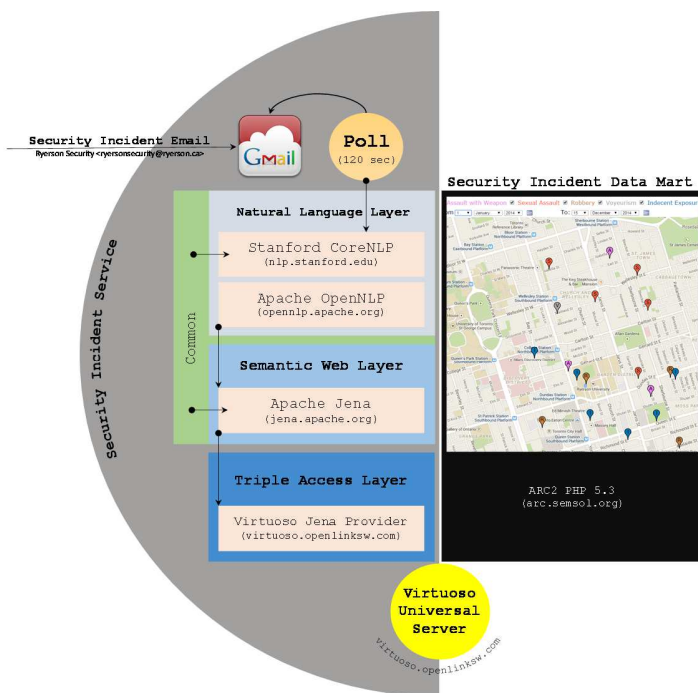


Figure 3. Machine Readable Security Incident Notification software system architecture. The left hemisphere is the software service which polls each 2 minutes a gmail account, seurityincidentsemanticweb@gmail.com, which is subscribed to be notified for any security incident by Ryerson University Integrated Risk Management. After fetching new incidents, the Natural Language Layer parse the text and extract information to an object of type `SecurityIncident`, defined in `Common`, and hand it to next layer. The Semantic Web Layer serialize the populated object to OWL individuals and pass them to Triple Access Layer to store them in the Virtuoso triple store. The right hemisphere is the primitive dashboard at <http://semionet.rnet.ryerson.ca/sio/> which shows the incidents geographical distribution filtered by the date of event.

To provoke SIO reuse, as a future work, we show that our work is endorsed by the Semantic Web community by registering our dataset to Linked Open Data [11] and adding it to the Linked Open Data Cloud.

A. Automatic Instance Data Population

We develop an infrastructure to transform textual notification of security incidents to machine readable representation in SIO. The software system, namely Machine Readable Security Incident Notification (MRSIN), has two main components, Security Incident Service and Security Incident Data Mart. Fig. 3. The former consists of I) Natural Language Layer (NLL) which extracts temporal and factual information from incident notification text to data objects, II) Semantic Web Layer (SWL) which serializes objects to Web Ontology Language individuals, `sio:SecurityIncident` in particular, and III) Triple Access Layer (TAL) which stores the individuals to the well-known triple store, Virtuoso Universal Server¹. All layers are in compliance with layering architecture and are working within a passive run on service. The later component incorporates a geographical distribution data mart of security incidents to support police and government with decisions, and criminologists with suggestions. This layer plays as the user interface to the system.

NLL is supposed to parse the textual content of the security incident report and extract entities and their property-values along with the co-references. Extracting temporal expression such as incident report date and event date, finding the victim and subject of the event and their associated property-values are the major tasks in this layer. There are toolkits for this kind of task among which the Apache OpenNLP² and Stanford

University Stanford Named Entity Recognizer (NER)³ are the most cited. However, these libraries working properly in general text corpus and will fail in our domain specific context. Therefore, we should train them (the model) to learn our textual paradigm. Since this task make a research shift to the current one, we leave it for future work and stay with the NER (7 class model trained: Time, Location, Organization, Person, Money, Percent, Date) along with some customizations

Java JDK 1.8 and PHP 5.5 are the programming languages for the service and data mart components respectively, NetBeans⁴ 8.0 is the Integrated Development Environment (IDE). We use Apache Jena⁵ in our SWL and Virtuoso Jena Provider⁶ for TAL.

B. Knowledge Extraction

The ontology allows users to semantically search and retrieve security incident information. Examples of semantic search scenarios may be: finding incidents with a specific type of security threat, retrieving sub-incidents of an incident, or searching incidents in which one particular suspect is involved. These queries can be expressed by SPARQL query language and be asked from our dataset SPARQL or JDBC endpoint. We show a SPARQL query example in Fig. 4 to search different types of security incidents in a date range. The sample result would be such in Fig. 5.

³ <http://nlp.stanford.edu/software/CRF-NER.shtml>

⁴ <https://netbeans.org/>

⁵ <https://jena.apache.org/>

⁶

<http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtJenaProvider>

¹ <http://virtuoso.openlinksw.com/>

² <http://opennlp.apache.org>

```

PREFIX sio: <http://ls3.rnet.ryerson.ca/ontologies/sio/>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX time: <http://www.w3.org/2006/time#>
SELECT distinct ?type ?lat ?lng ?year ?month ?day
WHERE
{
  GRAPH <http://ls3.rnet.ryerson.ca/SecurityIncident/test>
  {
    {?securityIncidentId rdf:type sio:Assault}
    UNION
    {?securityIncidentId rdf:type sio:AssaultWithWeapon}
    UNION
    {?securityIncidentId rdf:type sio:SexualAssault}
    UNION
    {?securityIncidentId rdf:type sio:Robbery}
    UNION
    {?securityIncidentId rdf:type sio:Voyeurism}
    UNION
    {?securityIncidentId rdf:type sio:IndecentExposure}
  }
  ?<p> <p> <o>
  ?securityIncidentId rdf:type ?type.
  ?securityIncidentId event:place ?placeId.
  ?placeId geo:lat ?lat.
  ?placeId geo:long ?lng.
  ?securityIncidentId event:time ?timeId.
  ?timeId time:inDateTime ?dateTime.
  ?dateTime time:year ?year.
  ?dateTime time:month ?month.
  ?dateTime time:day ?day.
  FILTER
  (
    (?type != owl:NamedIndividual) &&
    (
      1 = 1
    )
  )
}

```

Figure 4. Base query to search different types of security incidents in a date range

V. RELATED WORK

There are several similar initiatives to model event-focused concepts in knowledge representation [16], medical informatics [17] [18], sports [19], business news [20], scholarly events [21], life events [22] and multimedia community [23] [24]. In [17] the authors present Adverse Events Reporting Ontology (AERO) to report adverse event, any untoward medical occurrence in a patient or clinical investigation. [18] designs an event ontology for application in the machine understanding of infectious disease-related events reported in natural language text. Concepts and terminology in the field of sports events and their relationships are studied in [19]. Pattern-based approach is used in [20] to build *newsEvents* ontology to model business events, the affected entities and relations between them. [21] presents a very thorough process from creating ontology, automatic data instance population, and knowledge extraction interface in domain of scholarly events. We highly appreciate this work and mainly follow the same approach in our work.

Much closer to our work's domain we can name Crime Emergency Event Model (CE2M) [25]; although, despite what its name implies, it is an ontology for emergency events rather than crimes. Also, [26] constructs an event ontology to describe cyber crimes reflected in Web such as online fraud, internet pornography, illegal trade, false advertising, violations of privacy, network gambling, damage to reputation; it is still not a general crime domain ontology. [27] describes a knowledge base of Politically Motivated Violent Events (PMVE)

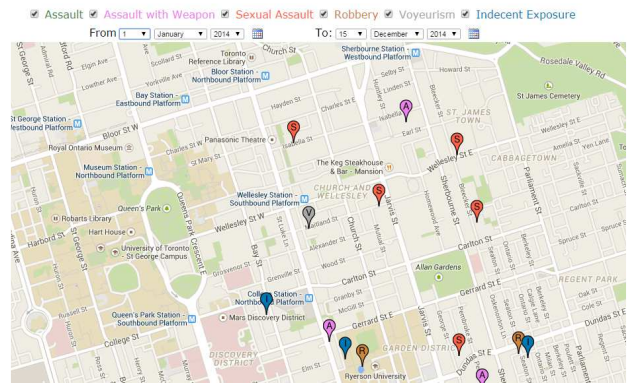


Figure 5. The snapshot of system dashboard at <http://semionet.rnet.ryerson.ca/sio/>. It locates the incidents geographically with its type color and capital letter from the query result in Figure . The shown icons are incidents within the 2014 year.

consisting of a domain ontology and of instance data. Sadly, this work does not publish its ontology specification.

Barring [20], all aforementioned works create their own event ontology for their domain of study, despite the fact that we have capable event ontologies. Event [9] and TimeML [35] are two cases based on different ontological view of the world. The latter employ linguistics and uses mark-ups to express event and time entities while the former prefer to focus on real happening of the event and comply with RDF/N3 representation. Event also reuse the Timeline [5] for temporal references in events. Timeline reuse, also, OWL-Time [6]. We respect reuse practice and not only adopt Event as the base ontology for security incidents instead of creating a new one, but also develop our SIO with regard to reuse principle.

VI. CONCLUSION

In this paper, we present Security Incident Ontology, SIO, a novel light-weight domain ontology for security incidents. We use Timeline ontology to express time references and adopt Event ontology to representing any security threats which raise the security alarm in a community. SIO is developed in OWL2 with Protégé by UPON ontology engineering methodology. SIO wants to present a unique ontological machine readable way to the security incident detectors to report security events. To reinforce the feasibility and capability of our work, we populating security incident notifications of Integrated Risk Management (IRM) at Ryerson University in SIO. To comply with the Semantic Web community principles we publish our dataset and provide SPARQL endpoint to extract knowledge. We hope that SIO wedge his way into the Web and obtain highest reuse rank to raise human and machine readability and common ubiquitous comprehension of security incidents.

ACKNOWLEDGMENT

The authors graciously acknowledge funding from the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

- [1] PublicEngines, "CrimeReports," PublicEngines, [Online]. Available: <https://www.crimereports.co.uk/>. [Accessed 17 10 2014].
- [2] Ryerson University, "Integrated Risk Management (IRM)," [Online]. Available: <http://www.ryerson.ca/irm>. [Accessed 17 10 2014].

- [3] Canadian Broadcasting Corporation, "Toronto Crime by Neighbourhood," [Online]. Available: <http://www.cbc.ca/toronto/features/crimemap/>. [Accessed 17 10 2014].
- [4] BRS Labs, "AI Sight," BRS Labs, [Online]. Available: <http://www.brslabs.com/index.html#aisight>. [Accessed 17 10 2014].
- [5] Y. Raimond and S. Abdallah, "The Timeline Ontology," Centre for Digital Music, Queen Mary, University of London, [Online]. Available: <http://motools.sourceforge.net/timeline/timeline.html>. [Accessed 17 10 2014].
- [6] J. R. Hobbs and F. Pan, "Time Ontology in OWL," The World Wide Web Consortium, [Online]. Available: <http://www.w3.org/TR/owl-time/>. [Accessed 17 10 2014].
- [7] J. R. Hobbs and F. Pan, "An Ontology of Time for the Semantic Web," ACM Transactions on Asian Language Information Processing (TALIP), vol. 3, pp. 66-85, 2004.
- [8] J. R. Hobbs, "OWL-Time (formerly DAML-Time)," [Online]. Available: <http://www.isi.edu/~hobbs/owl-time.html>. [Accessed 17 10 2014].
- [9] Y. Raimond and S. Abdallah, "The Event Ontology," Centre for Digital Music, Queen Mary, University of London, [Online]. Available: <http://motools.sourceforge.net/event/event.html>. [Accessed 17 10 2014].
- [10] T. Heath, "Linked Data - Connect Distributed Data across the Web," Linked Data community, [Online]. Available: <http://linkeddata.org/>. [Accessed 17 10 2014].
- [11] M. Schmachtenberg, C. Bizer, A. Jentzsch and R. Cyganiak, "The Linking Open Data cloud diagram," [Online]. Available: <http://lod-cloud.net/>. [Accessed 17 10 2014].
- [12] Ryerson University, "Ryerson Security Incident Records," [Online]. Available: http://www.ryerson.ca/irm/alerts_reports/alerts/index.html. [Accessed 17 10 2014].
- [13] Wikimedia Foundation, "Toronto," Wikimedia Foundation, Inc., [Online]. Available: <http://en.wikipedia.org/wiki/Toronto>. [Accessed 17 10 2014].
- [14] Toronto Police Service, "Toronto Police Service Mailing Lists," [Online]. Available: <https://secure.torontopolice.on.ca/tpsml/>. [Accessed 17 10 2014].
- [15] Toronto Police Service, "TPS Crime Statistics," [Online]. Available: <http://www.torontopolice.on.ca/statistics/>. [Accessed 17 10 2014].
- [16] J. F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks/Cole, 1994.
- [17] M. a. B. R. R. a. R. A. Courtot, "Reporting Adverse Events: Basis for a Common Representation.," in ICBO: International Conference on Biomedical Ontology, Buffalo, NY, USA, 2011.
- [18] A. Kawazoe, H. Chanlekha, M. Shigematsu and N. Collier, "Structuring an Event Ontology for Disease Outbreak Detection," BMC bioinformatics, vol. 9, p. S8, 2008.
- [19] J. Xiao and J. Chen, "Features, Improvements and Applications of Ontology in the Field of Sports Events During the Era of the Semantic Web," in Chinese Lexical Semantics, Springer, 2013, pp. 718-727.
- [20] U. Losch and N. Nikitina, "The newsEvents Ontology: An Ontology for Describing Business Events," Citeseer, 2009.
- [21] S. Jeong and H.-G. Kim, "SEDE: An Ontology for Scholarly Event Description," Journal of Information Science, 2010.
- [22] I. Trochidis, E. Tambouris and K. Tarabanis, "An Ontology for Modeling Life-events," in IEEE International Conference on Services Computing, 2007.
- [23] S. Abdallah, Y. Raimond and M. Sandler, "An Ontology-based Approach to Information Management for Music Analysis Systems," 2006.
- [24] Y. Raimond, T. Gängler, F. Giasson, K. Jacobson, G. Fazekas, S. Reinhardt and A. Passant, "The Music Ontology," [Online]. Available: <http://musicontology.com/>. [Accessed 17 10 2014].
- [25] W. Wang, W. Guo, Y. Luo, X. Wang and Z. Xu, "The Study and Application of Crime Emergency Ontology Event Model," 2005.
- [26] L. Cunha, H. Yun and Z. Zhaoman, "An Event Ontology Construction Approach to Web Crime Mining," in Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2010.
- [27] P. O. Wennerberg, H. Tanev, J. Piskorski and C. Best, "Ontology Based Analysis of Violent Events," in Intelligence and Security Informatics, 2007 IEEE, 2007.
- [28] R. Mizoguchi, "Tutorial on Ontological Engineering Part 2: Ontology Development, Tools and Languages," New Generation Computing, vol. 22, pp. 61-96, 2004.
- [29] A. a. M. M. a. N. R. De Nicola, "A proposal for a unified process for ontology building: UPON," in Database and Expert Systems Applications, 2005.
- [30] Stanford Center for Biomedical Informatics Research, "Protege Ontology Library," Stanford Center for Biomedical Informatics Research, [Online]. Available: http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library. [Accessed 17 10 2014].
- [31] J. F. Allen and G. Ferguson, "Actions and Events in Interval Temporal Logic," Journal of Logic and Computation, vol. 4, pp. 531-579, 1994.
- [32] W3C Semantic Web Interest Group, "Basic Geo (WGS84 lat/long) Vocabulary," W3C Semantic Web Interest Group, [Online]. Available: <http://www.w3.org/2003/01/geo/>. [Accessed 18 10 2014].
- [33] A. Einstein, "The Relativity of Simultaneity," in Relativity: The special and general theory, New York: Henry Holt and Company, 1920.
- [34] T. Berners-Lee, "Linked Data," The World Wide Web Consortium (W3C), [Online]. Available: <http://www.w3.org/DesignIssues/LinkedData.html>. [Accessed 19 10 2014].
- [35] J. Pustejovsky, J. M. Castano, R. Ingria, R. Sauri, R. J. Gaizauskas, A. Setzer, G. Katz and D. R. Radev, "TimeML: Robust Specification of Event and Temporal Expressions in Text," New directions in question answering, vol. 3, pp. 28-34, 2003.

CARP: Correlation-based Approach for Researcher Profiling

Hassan Nouredine
EDST
Lebanese University
Beirut, Lebanon
HES-SO/FR
Fribourg, Switzerland

Iman Jarkass
IUT
Lebanese University
Saida, Lebanon

Hussein Hazimeh*
HES-SO/FR
Fribourg, Switzerland

Omar Abou Khaled
HES-SO/FR
Fribourg -Switzerland

Elena Mugellini
HES-SO/FR
Fribourg –Switzerland

Abstract—The accelerating progress in science with the active role of the communication media – mainly the web – make person in front of a difficult task, in finding appropriate information during a brief time. In a narrower context, many researches were created in the expertise retrieval domain, as an interesting and complicated task for the scientific community, in face of this huge amount of data scattered across the web. Benefiting from the semantic web technologies and the efforts of data structuring, in this paper we propose a novel approach of correlation based profile building, by exploiting heterogenous web sources. The aim is to generate comprehensive and validated profiles about researchers and experts in the computer science domain.

Keywords- *Expertise Retrieval; Profile Matching; Profiling; quality of data; Semantic Web*

I. INTRODUCTION

Since the web has established, it is still growing in a rapid manner. Where, every second millions of bytes are added around the world. Inconsistent with this growth, many web technologies have been emerged and participate in enhancing the efficiency of the web, like semantic web technologies. Therefore this massive growth forms the main motive to use web as a rich source of information and interactions. But at the same time, create more complex problems in the information retrieval domain. For instance, extracting specific and accurate web information must take into consideration the problems of conflicted, repeated and outdated data. In this context, the essential role that played by the web in the scientific progress, make the scientific community interested to solve this problem, especially in the profiling and expertise retrieval domains [1,8].

In this paper we proposed a Correlation based Approach for Researcher Profiling: CARP. The profiling task is going worsen with this massive and scattered amount of increased information across the world of web. As we proposed that we are going to cover the part of the problem relating to researcher profiling. The problem can be briefed as follows: if someone wants to search for a profile related to a specific researcher X, this will be a time-consuming process, especially there are no such standard sources that contain confirmed-content researchers' profiles. Even if we can find many systems as in [8,2,12] and others that provide scientific information related to researchers in several domains. However these data are still lacking to the quality in several cases. Cases lack such researchers' information, and others contain conflicted or outdated ones. Therefore we propose a new profiling approach

based on correlating information from heterogeneous web sources, which contain confirmed data about researchers. The objective is to overcome the quality of data issue, and provide comprehensive and validated information about researchers, passing through a matching procedure.

In the rest of paper, the proposed approach is described as follow: The section 2 reviews and discusses the related work. The next section gives an overview on the proposed approach and describes the system architecture. The section 3 presents the obtained results, which are evaluated in the section 4. Finally, the paper is concluded in the last section.

II. RELATED WORKS

This section is composed of two parts. The first one mentions and explains the recent approaches in expert finding, and the other lists the latest approaches related to profile matching among multiple web resources.

A. Expert Finding Systems

The approaches submitted in this area are dealing with finding experts, where the most critical issue is what sources they are going to choose to find experts and create their profiles. The most popular system is Arnetminer, this system is based on finding and creating experts profiles in computer science domain and represents them semantically [8]. Microsoft Academic Search, another expert finding system, offers a diversity of functions for searching experts in several domains of sciences [2]. Other systems like INDURE, are limited to a set of organizations or universities, it provide functions for exploring profiles across these organizations in multiple disciplines [1,3]. The majority of the mentioned systems operate by extracting information from a single source, and even if some use multiple sources, they focus on a single source as the principal one compared to other sources. For example, Arnetminer is based on the home pages as source to extract the basic profile attributes, and then complete the profiles with the information extracted from DBLP [14]. While, our approach is to apply the concept of correlation between multiple web sources, leading to merge the discarded information in a unified profiles. Therefore, we consider that each source has his separate profiles, and all profiles from different sources must pass through a profile matching stage.

B. Profile Matching

Many approaches have proposed in this context and each one address this issue from his perspective, in this section we

will focus on those who concentrate on web and social networks as a main source of information. In some approaches as in [4], they address this problem at the level of only two social networks, also they suppose that we have only one person profile among each social network, this approach and others use machine learning algorithms to resolve their decisions regarding the matching process. In [5], they proposed an expert finder system based on semantic matching between user profiles, they use the process of spreading to include additional related terms to a user profile by referring to an ontology (Wordnet or Wikipedia) [5]. Jain, Kumaraguru and Joshi [6] proposed an approach that matches profiles across Facebook and Twitter, by exploiting syntactic and image matching methods to discover the similarity between user profiles. In [7], they propose a vector based comparison algorithm that computes the similarity between two profiles according to their vector of attributes, and then classify whether they are the same or not based on a specific threshold. The mentioned approaches solve the problem partially. On the one hand they always apply the correspondence between social networks that are similar and almost have the same profile attributes. On the other hand they ignore the problem of name disambiguation by assuming that there is a unique profile for each person in different social networks. In contrast, we are working on matching profiles between multiple sources with different types, and we consider also the problem of name disambiguation by investing the detected similarity between profiles, as described in the next section.

III. PROPOSED APPROACH

Our proposed approach CARP is aiming to find a solution that addresses the problem of researchers' profiling, by benefiting from the heterogeneity of structured and unstructured data distributed across the web, this will be carried out through a complete architecture composed of six main components as illustrated in figure 1.

Problem Definition and formulation: the main goal of CARP approach is to produce researchers' profiles by correlating information coming from several web resources. Let R_i be a specific web source (DBLP, MAS, LinkedIn), contains a set of profiles that belongs to a specific author name: $R_i = \{P_1, P_2, \dots, P_n\}$, and each profile P_j contains a set of attributes $P_j = \{A_1, A_2, \dots, A_n\}$, where R_i, P_j, A_k is a specific attribute for a profile that belongs to a specific web resource. The aim is to find similar profiles among these sources by matching information extracted from their attributes, and then merge this information to produce complete profiles.

A. Ontologies

The initial stage in our architecture is to construct the system ontology. It covers all classes and properties describing the researchers' profiles, their relationships and their scientific products. It supports and facilitates the information extraction and storage processes. Our ontology is based on the SWRC ontology (Semantic Web for Research Communities) [13], and it is composed of four major classes: the class person, document, education, position and organization, where each person (researcher) has a set of object and data properties. For instance a researcher has an education (PhD, Master, Bachelor), or he is an author for a document.

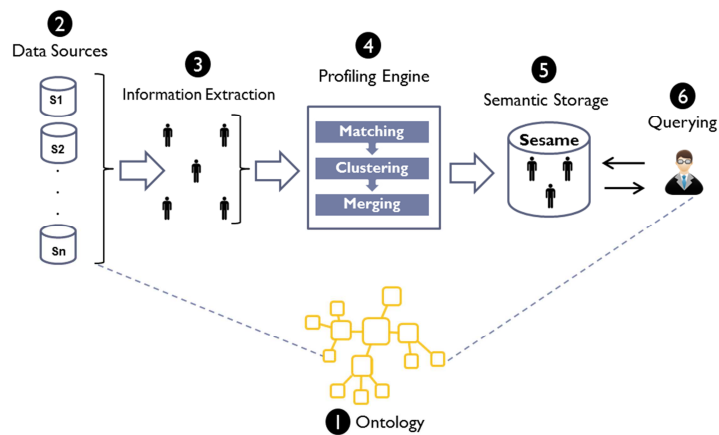


Figure 1. Architecture Components

B. Data Sources

Since our proposal is to apply the concept of correlation between multiple web sources, we have analyzed the data granted from different web sources. Then, we decided to use two types of sources: The bibliographic sources and the social networks. On the one hand, bibliographic sources provide essential information about researchers, their scientific activities and publications. On the other hand, there is a big trend to use social networks and especially professional networks. In this context we have chosen MAS, DBLP [2,14] as bibliographic sources and LinkedIn [9] as a social network.

C. Information extraction

The system starts operations with the structured information extraction, where the provided information are granted by the API of each source. However due to the limitation of the provided structured information, our system also extracts information from unstructured text, from home pages, publications and biographies. Two methods are used for this task. The first one is GATE (General Architecture for Text Engineering) as rule based method. It is used to extract the existing contact information (affiliation, email and location) from the publications headers. Thus GATE is suggested because it shows an average precision and recall of 90-95% on extracting contact information [10]. The second method is CRF (Conditional Random Fields), this method is employed to extract other attributes (education and the list of historical positions) from biographies (education existed in publications, homepages and LinkedIn profiles), by tagging them based on a built training set. We decide to use CRF, because it has lowest error rates for POS tagging compared to other methods [11]. Based on the chosen methods, the extraction process produces a set of preliminary profiles, presenting the attributes available in each source.

D. Profiling Engine

The main goal of this engine is to generate unified and confirmed profiles, passing through a correlation between the preliminary profiles. The correlation process is composed of three steps: matching, clustering and merging, as shown in the figure 2. The profile engine starts operating firstly with the matcher M1 that aimed at finding the similarity between

profiles from DBLP and MAS. We decide to use these two sources according to the permanent availability of two common profile attributes (affiliation and publication title), and to achieve this we have employ two string matching algorithms. We chose Jaro-Winkler (1) to calculate the similarity between affiliations, because Jaro-Winkler metric seem to be intended primarily for short strings (e.g., personal names), and Jaccard index (2) to calculate the similarity between publication titles.

$$Sim_{Jaccard}(R_1.P_2.A_2, R_2.P_3.A_2) = \frac{R_1.P_2.A_2 \cap R_2.P_3.A_2}{R_1.P_2.A_2 \cup R_2.P_3.A_2} \quad (1)$$

$$Sim_{JW}(R_1.P_2.A_2, R_2.P_3.A_2) = Sim_{Jaccard}(R_1.P_2.A_2, R_2.P_3.A_2) + \frac{p}{10} (1 - Sim_{Jaccard}(R_1.P_2.A_2, R_2.P_3.A_2)) \quad (2)$$

Let S_a be the similarity result of matching between two affiliations where $S_a = Sim_{JW}(R_1.P_i.affiliation, R_2.P_i.affiliation)$. S_p is the similarity result of matching between two publications titles where $S_p = Sim_{Jaccard}(R_1.P_i.Publication, R_2.P_i.Publication)$, and S_c is the similarity result of matching between two coauthors titles where $S_c = Sim_{JW}(R_1.P_i.coauthor, R_2.P_i.coauthor)$. Additionally, t_a , t_p and t_c are the threshold numbers, which represent the percent of matching between affiliations, publications and coauthors respectively, where $S_a \geq t_a$, $S_p \geq t_p$ and $S_c \geq t_c$. The matching process between each profile from DBLP with each profile from MAS starts by comparing the list of publications, if the number of matched publications $\geq t_p$ we decide that the two profiles belong to the same entity, else we continue the matching process by comparing the rest of publications using affiliation extracted from each publication, if the matching remains null we resolve the similarity based on the coauthors attribute. For each set of matched profiles we create a cluster C_i , and populate each matched profile to its parent cluster. After obtaining set of clusters, each cluster must undergoes to a merging operation, this step aims at unifying the set of profiles in each cluster into one profile and validate its attributes by applying several merging rules for each attribute.

After finishing the first correlation by matching (M1), clustering and merging between DBLP and MAS, we obtain a set of unified profiles. These profiles will act as an input for the matcher M2 that aim to complete the profiling operation by complementing the rest of profile attributes from LinkedIn. Each unified profile will be matched by a set of LinkedIn profiles using three attributes: affiliation, publication and education. The matching process starts by comparing publication titles, if there is common publication between two profiles we decide that the two profiles belong to the same person, else we compare the affiliations if there are the same we decide that the two profiles are for the same person, else we compare the list of education organizations to detect the similarity between two profiles. In this case, deciding whether two profiles belong to the same person will be easier, because the LinkedIn data are typed by the users themselves, and consequently there is an absence of the name disambiguation problem. This is the reason to not repeat the clustering method, and merging directly the matched profiles into the final unified profiles (the output) as shown in the figure 2.

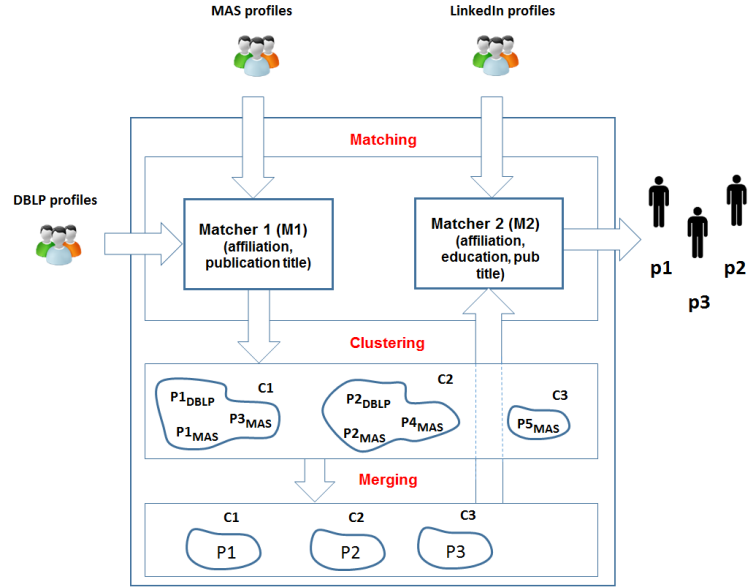


Figure 2. Profiling engine steps

E. Semantic storage

The Semantic Web technologies enhance the ability to discover relations between properties more than current traditional databases, thus we propose to store the extracted profiles in a semantic database in form of RDF triples according the system ontology.

F. Querying

The final step in our architecture is to retrieve information about researchers, where the query will be a researcher name. The query language used for this task is SPARQL.

IV. PERFORMANCE EVALUATION AND RESULTS

Referring to the architecture described in figure 1, we have implemented the various system elements, and thus provided web interface for receiving user requests and respond with relevant results. The prototype of our architecture is implemented using JavaEE, where all the tests are performed on Intel 2.93 core i7, 8GB of RAM PC.

Figure 3. An example of researcher profile

The figures 3 present an example of profile generated by the system. We can see the benefit of the correlation, mainly by

obtaining comprehensive and confirmed profile, with attributes retrieved from various sources. The obtained results show that the same attribute is not always recovered from the same source, so that the missed attribute from some source can be provided in the other. This increases the possibility of retrieving information. For instance, the attribute “summary” are extracted from LinkedIn, and in case of absence, it can be extracted from the biography inside publications.

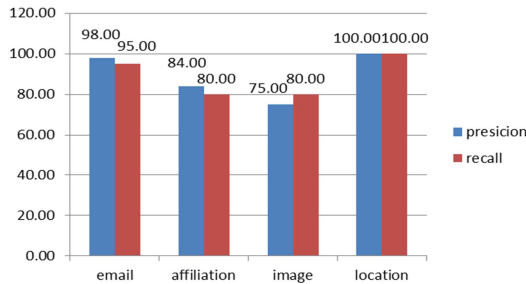


Figure 4. Precision and recall measures for each attribute

The figure 4 presents the precision and recall for four different attributes among 25 tested profiles, which they form the principal profile attributes. Based on these measures we are analyzed the results, thus the attributes “email” and “affiliation” are extracted from publication using GATE. Hence the strength of the precision and recall depends partially on the accuracy of GATE, and as we observe that we are obtaining 98 percent and 84 percent for email and affiliation respectively. The affiliation is sometimes failed to be extracted by GATE because of some problems. For instance, the language in which the affiliation is written, however in our case we are considering only the English language. The attribute “image” is extracted from three different resources: biographies, LinkedIn and MAS, resulting a precision of 75 percent. However we still need a strong face recognition method to validate the correctness of this attribute. Finally, the attribute “location” is extracted from two different sources publications and LinkedIn, this attributes has 100 percent of both precision and recall because location names are easy to be validated due to their limitation unlike affiliation and other attributes.

Additionally, the study of the availability of each attribute before and after the correlation has proved the efficiency in increasing it, especially for the attribute not strongly available. For instance the “image” attribute as shown in the Table I.

TABLE I. AVAILABILITY OF IMAGE ATTRIBUTE BEFOR AND AFTER CORRELATION

	Image from biography(publication)	Image from LinkedIn	Image from MAS
Before	53%	38%	57%
After	89%		

On another side, our approach was able to address the issue of name disambiguation in a low proportion, by benefiting from the partitioning of profiles among resources, which allows us to detect the diversity between profiles. Table II shows four profiles with name disambiguation tested between DBLP and

MAS. This issue is directly affected by the resolution rate of this problem by each source. In LinkedIn, it does not exist because users enter information by themselves. In MAS the problem is opposed, where we can find several profiles for the same researcher. Therefore the problem must be resolved in DBLP, where the disambiguation exists in various cases.

TABLE II. NAME DISAMBIGUATION RESULTS TESTED ON FOUR DIFFERENT AUTHOR NAMES

Author name	Num. of MAS profiles	Num. of DBLP profiles	Actual Num. of profiles	Num. of profiles after merging
Kai Eckert	4	2	2	2
Hong Shen	11	1	4	3
Michael Wagner	1	3	12	4
Feng Liu	1	1	4	2

V. CONCLUSION AND FUTURE WORK

In this paper we present CARP approach, which based on the concept of correlating information from several web resources, to satisfy the production of qualified profile information, our investigated approach shown promised results. Moreover, this approach has overcome the problem of name disambiguation in some cases by benefiting from the variety of profiles among the different sources. However we still need a strong approach addressing this problem, and as a future work, we propose to add a name disambiguation block aiming to split the target profiles before the correlation.

REFERENCES

- [1] K. Balog, Y. Fang, M. de Rijke, P. Serdyukov and L. Si. Expertise Retrieval.(2012)
- [2] (2014,July). Microsoft Academic Search [Online]. Available: <http://academic.research.microsoft.com/>.
- [3] E. A. Jansen. A Semantic Web based approach to expertise finding at KPMG.(2010).
- [4] Raad, E., Chbeir, R., Dipanda, A. User Profile Matching in Social Networks. In NBiS, 2010.
- [5] Thiagarajan, R., Manjunath, G. and Stumptner, M. Finding experts by semantic matching of user profiles. Technical Report, HP Laboratories. (October 2008).
- [6] P. Jain, P. Kumaraguru, and A. Joshi. @ i seek ‘fb.me’: identifying users across multiple online social networks. IW3C2, 2013.
- [7] Vosecky, J.; Hong, D.; and Shen, V. Y. User identification across multiple social networks. In Int. Conference on Networked Digital Technologies (2009).
- [8] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In SIGKDD 2008.
- [9] (2014,July). LinkedIn [Online]. Available: <https://www.linkedin.com/>.
- [10] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: An architecture for development of robust HLT applications. In ACL, 2002.
- [11] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Eighteenth international Conference on Machine Learning, 2001.
- [12] (2014,July).Google scholar. Available: <https://scholar.google.com>
- [13] (2013,July) The Semantic Web for Research Communities Ontology (SWRC). [Online]. Available: <http://ontoware.org/swrc/>
- [14] (2014,July). DBLP XML dataset [Online]. Available: <http://dblp.uni-trier.de/xml/>

APRImora: A Semantic Architecture for Patterns Reuse

Angélica Aparecida de Almeida Ribeiro¹, Jugurta Lisboa-Filho¹, Lucas Francisco da Matta Vegi¹, Alcione de Paiva Oliveira¹, Regina Maria Maciel Braga Villela² and Emílio José de S. Fonseca¹

¹ Departamento de Informática
Universidade Federal de Viçosa
Viçosa-MG, Brazil, 36570-000

² Departamento de Ciência da Computação
Universidade Federal de Juiz de Fora
Juiz de Fora-MG, Brazil, 36036-900

angelica.ribeiro@fagoc.br, jugurta@ufv.br, lucas.vegi@ufv.br, alcione@dpi.ufv.br, regina@acessa.com, emiliojsf@gmail.com

Abstract— Software patterns are computing artifacts used to document knowledge that may be reused during software development process. There are several types of patterns, such as analysis, design, and architectural, among others. Design patterns are the most well known by designers, but many of them are described in books and scientific papers, a recurring way of documenting patterns that limits their reuse potential. Aiming to not only minimize this limitation but also provide ways of recovering contextualized knowledge in these patterns, the present paper presents the architecture APRImora, an extension based on Semantic Web of the Analysis Patterns Reuse Infrastructure (APRI). In this architecture, the patterns are documented by metadata defined as application profiles of the Dublin Core standard and stored in the RDF format, allowing them to be discovered by search engines. The APRImora architecture helps designers discover and reuse software patterns based on semantic relations, which favors their dissemination and reuse.

Keywords- *semantic Web; analysis pattern; design patterns; retrieving information; pattern repositories.*

I. INTRODUCTION

Reuse mechanisms such as design patterns [8] and analysis patterns [7] are crucial computing artifacts in the software development process. However, the great challenge a designer still faces today is discovering a pattern that solves a given problem. Search engines not always help because, although the search is facilitated by its constant improvement, a simple query may return a large number of results, many not relevant for the context. For example, when searching on Google for the design pattern Singleton typing only the word Singleton, with no additional information, approximately 6.6 million results are returned, a great number of which irrelevant and inappropriate to solve the problem.

According to Cunha et al. [5], web data are organized to be read and understood by humans and not by software agents. This is due to the way the Web organizes the information available on the Internet by using hypertext provided by the HTML language. Its organization allows for a more user-friendly interaction with the global network, however, the HTML language is used only to list resources to the user with no meaning presented or association on the meanings.

In order to solve this issue, the Semantic Web was created, which aims to transform the Web of documents into the Web of data, in which all data on the Web must be associated to a given

semantics and are connected among themselves. In the Web setting, this would represent attributing meaning to the data, connecting them with other datasets or knowledge domains, and creating a relation of meaning among the content published on the Internet so that they can be analyzed not only by humans, but also by software agents [5]. That results in more effective searches with results closer to what the user wants.

In order for computing artifacts to be easily retrieved and reused, Vegi et al. [20] proposed the APRI, i.e., Analysis Patterns Reuse Infrastructure. Its goal is to allow the analysis patterns to be documented, disseminated, and enhanced through an approach based on metadata and Web services. The analysis patterns added to the APRI are documented by an application profile of the Dublin Core metadata standard and stored in the RDF format so that they are accessible to search engines [21].

The present paper describes a proposal for extending the APRI architecture for an architecture based on the Semantic Web, turning it into a Patterns Reuse Semantic Infrastructure called APRImora. The remaining of the paper is structured as follows. Section 2 presents a review of the Semantic Web, besides other technologies related to the research. Section 3 lists some correlated studies. The architecture proposed for APRImora is presented in section 4. The final considerations and some future studies are described in section 5.

II. THEORETICAL FRAMEWORK

A. Semantic Web

According to Berners-Lee et al. [1], the Web developed more quickly as a medium for documents to the people than to the data and information that can be automatically processed. Consequently, most times a search is conducted, thousands of results with little or no relevance are obtained. This occurs because the meaning of the Web content cannot be processed by machines, besides the ability of software to interpret sentences and extract useful information for the users being limited.

From this context appeared the Semantic Web, which makes up for this issue by providing means of organizing data and allowing them to be interpreted by computers. The Semantic Web resources allow search results to be improved and facilitates the automation of relevant tasks. Berners-Lee et al. [1] add that the Semantic Web is not a separate Web, but an extension of the current one.

The challenge of the Semantic Web is to provide a language that expresses both the data and the rules to think such data, and that allows the rules of any knowledge system to be exported to the Web [1]. Figure 1 illustrates the layers that make up the Semantic Web. Details on each layer can be found in [2].

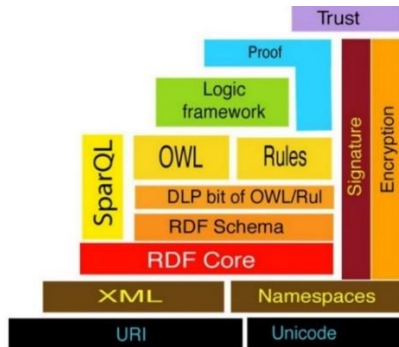


Figure 1. Architecture of the Semantic Web [2]

B. SPARQL Protocol and RDF Query Language

RDF is a flexible and extensible format used to represent information on Web resources such as personal information and metadata on digital artifacts such as music and images, among others. Moreover, it provides means to integrate different sources of information by means of a set of triples, formed by the concepts of subject, predicate, and object [10].

SPARQL Protocol and RDF Query Language (SparQL) is a language and a data-access protocol developed and recommended by the W3C for queries on RDF files [9].

C. Linked Data

The goal of the Semantic Web is to provide means to organize data and enable its interpretation by computers, thus making work more helpful, besides developing systems that can support reliable interactions on the network [22]. Moreover, the Semantic Web is seen as a layer of the Web in which one can publish, obtain, and use data that may be directly or indirectly processed by machines [22].

However, its importance goes beyond only making data available on the Web since it connects such data so that a person or machine is able to explore them. This interconnection of data allows for more information related to a given piece of data searched to be found [3]. It provides means for the conventional Web to be transformed into a Web with machine-processable data. This connection is called Linked Data.

Linked Data refers to the set of best practices to publish and connect data on the Web. These practices led to the creation of a global data space that contains billions of pieces of information, the so-called Web of Data [4]. As with the hypertext Web, the Web of Data is built from Web documents [3]. However, the difference between them is that on the hypertext Web the related links are anchored in documents written in HTML. On the Web of Data, arbitrary things are connected by the description in RDF using Uniform Resource Identifiers (URIs), which identify any type of object or concept.

The URI provides a simple and extensible way of uniquely identifying through a string of characters an abstract or physical resource [2].

D. Web Service

Web service is a solution used in systems integration and communication between different applications, with the advantage of standardized communication between services, which allows the platform and programming language to be independent, such as a system developed with Java running on a Linux server being able to seamlessly access a service implemented on .NET running on a Microsoft server [15].

Semantic Web Service (SWS) is an intersection of the Web services and Semantic Web. According to Martin et al. [12], the main focus of SWSs is the use of richer and more declarative descriptions of elements in the distributed computing dynamics. These descriptions enable complete automation, more flexible service, the use and construction of more powerful tools.

E. Dublin Core Metadata Standard

The set of elements of the Dublin Core metadata standard is composed by 15 basic and generic elements, which allows it to be used to describe a wide range of resources [6].

Although it is a simple metadata standard that doesn't offer enough resources to describe data of complex domains, Dublin Core's versatility in documenting artifacts of several types is achieved by extensions known as application profiles. Since 2000, the Dublin Core community has proposed profiles to adequate the elements to generate new metadata sets to meet the specific requirements from different domains.

In the APRI context, two Dublin Core metadata profiles were defined: DC2AP [18] to document analysis patterns and DC2DP [17] to document design patterns.

The DC2DP elements were defined from the combination of the Dublin Core metadata standard elements, the elements of the template by Gamma et al. [8], and the DC2AP elements. Dublin Core is the basis of DC2DP, so all elements belonging to this metadata standard are part of the DC2DP profile. The template by Gamma et al. contributes with the specific elements to document design patterns, while the version control elements were reused from the DC2AP profile.

Similarly to DC2AP, DC2DP is also machine processable, allowing design patterns to be described and published as Linked Data through RDF files, therefore the patterns can be retrieved and reused more precisely by a computer. A detailed technical description of these two application profiles can be found at <http://www.dpi.ufv.br/projetos/apri>.

III. RELATED WORK

Monteiro [13] proposes an architecture based on Semantic Web for digital educational repositories in healthcare that make available learning objects with educational aspects described in metadata forms. Learning objects (LO) are structured resources used to make self-learning content available and are appropriate to develop Distance Learning content. Making LO available on the Web contributes to lower costs with their production. In order to allow LOs to be available, Learning Management System (LMS) collections are created so that the LOs are accessed from the course environments themselves. Digital Educational Repositories (DER) can also be developed, where an increasingly larger number of objects is shared. Hence, the

DER must have an appropriate architecture and carry enough descriptive information for the selection of LOs, particularly regarding educational aspects.

Thus, Monteiro [13] proposes adding technologies to the DER architecture capable of refining the search result in LO by using the Semantic Web as a key element to contribute to the search refinement.

Another study directly related with the proposal presented in the current paper is the Analysis Patterns Reuse Infrastructure (APRI) [20]. APRI's architecture is inspired by the Spatial Data Infrastructures (SDIs), which, in turn, are defined as collections of technologies, policies, and institutional arrangements that facilitate the availability of and access to spatial data [14]. Many SDIs employ the concept of service-oriented architecture (SOA), thus allowing shared, distributed, and interoperable environments to be developed based on Web services.

Similarly to the SDI, APRI uses metadata to describe the analysis patterns to be stored in its repository. Its goal is to provide mechanisms for the analysis patterns to be made available and found by designers, which helps in these patterns being publicized and, consequently, reused.

IV. APRIMORA: AN ARCHITECTURE BASED ON SEMANTIC WEB FOR PATTERN REUSE

Initially, the APRI was developed focusing only on the reuse of analysis patterns [19]. Later, Ribeiro et al. [17] proposed adding design patterns to the APRI repositories, thus broadening its application scope and allowing the patterns to be reused not only during the software development analysis, but also during the design stage.

The patterns stored in the APRI repository can be retrieved from a Web browser and also through services that allow the patterns to be discovered, catalogued, and reused from the Pattern Portal (Figure 2). However, since search engines are used, the system may return a number of results that are irrelevant to the research context and inappropriate to meet the needs of the search.

The patterns documented in the APRI are described as Linked Data, which helps in their retrieval. Nevertheless, the metadata still have a limited formal semantics. The most common method to search for a dataset consists in searching for the elements using keywords that match the metadata.

The fact that metadata-based searches return a large amount of results outside the research context when using search engines requires the elements to be semantically defined. That gives the meaning of the elements and is a key point to help recover information from the repositories precisely. Thus, the present study extends the APRI to an architecture based on the Semantic Web able to help the user better recover patterns contained in its repositories, which originates APRImora, a Pattern Reuse Semantic Infrastructure (Figure 3).

APRI-mora architecture's core consists of the same components of the APRI, but with semantics added to the elements and to the way they are retrieved and inserted into the

structure. The following subsections describe the components of this architecture.

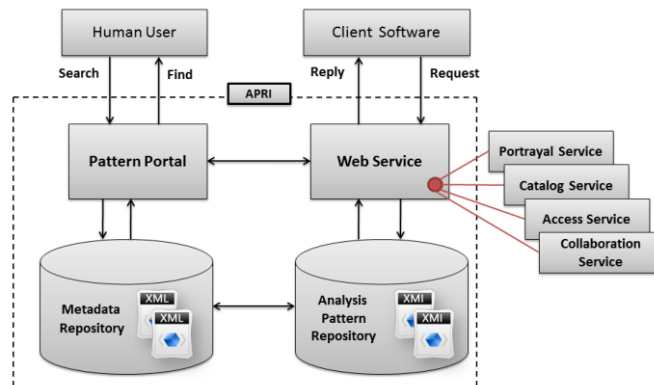


Figure 2. APRI architecture

A. Pattern Portal

The Pattern Portal is a portal composed of a set of websites focused on obtaining patterns. It provides tools and services for the discovery, cataloguing, and reuse of these patterns. Also available in the Pattern Portal is the metadata profile editor for analysis patterns (DC2AP Metadata Editor) [16] and the profile editor for design pattern metadata (DC2DP Metadata Editor).

Each element part of the metadata set of these two editing tools has rules regarding its enforcement, occurrence, and value type, which must be validated for the documented pattern to be consistent. After documenting the analysis or design pattern, the enforcement rules referring to these metadata are validated by a Web service called Validation Service. This service verifies, for example, if all fields whose occurrence rule is Mandatory have been filled. If not, the user will receive an error message.

The occurrence rules, regarding the plurality of the elements, are validated by the interface itself using buttons that add fields to elements that must be multiple while omitting simple fields.

Some elements are filled and validated by controlled vocabularies, i.e., sets of standardized terms that aid the data input and output in an information system, thus promoting greater precision and effectiveness in the communication between users and the information system [11]. Examples of controlled-vocabulary elements include type, format, language (DC2AP), and type and language (DC2DP).

When the Validation Service returns an affirmative response, the RDF Service can be used to generate an RDF document of the new pattern, which is automatically stored in the Metadata Repository in a user-defined folder.

Besides the metadata editing tools, the Apache Jena framework¹, is part of the Pattern Portal. Jena is a free, open-source Java framework to build Semantic Web and Linked Data applications. This framework is made up of different APIs interacting among themselves to process RDF data.

In the APRImora architecture, this framework is used to retrieve information from the data and metadata repositories, along with SparQL language, using the ARQ module, a query mechanism used by Jena that supports the SparQL language.

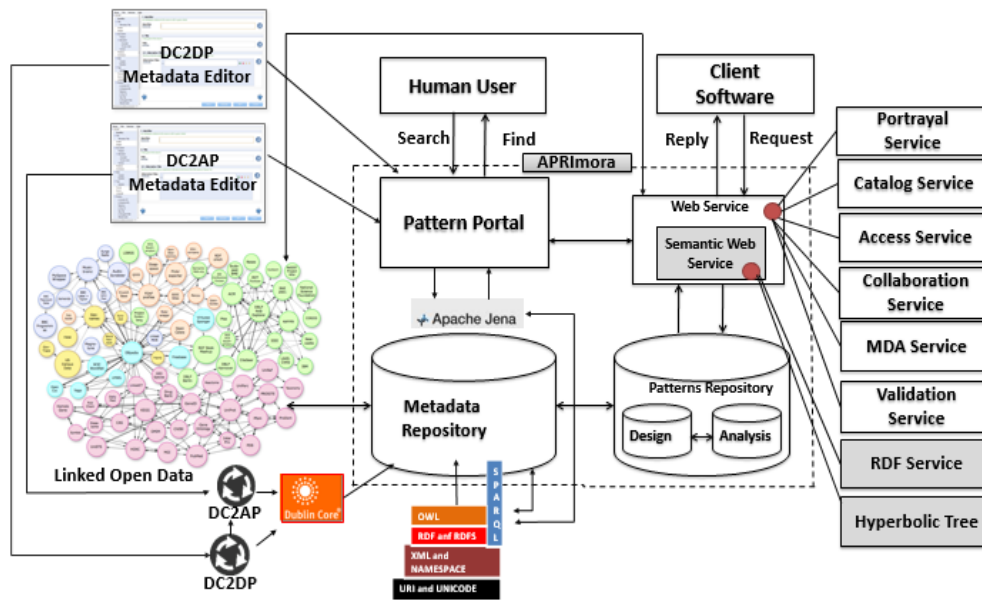


Figure 3. APRImora architecture

B. Semantic Web Layers

The URI, XML and Namespace, RDF and RDFS, and OWL technologies, belonging to the Semantic Web layers, are used directly in the documents contained in APRImora’s Metadata Repository. Other Semantic Web layers (Figure 1), such as Logic Framework, Proof, Trust, Signature, and Encryption, were not added to APRImora’s structure since there are still no recommendations by the W3C to use these layers [13].

The URI layer is used to assign a unique name to the elements using the Unicode standard, which is the universal standard adopted for this addressing. The URI guarantees the uniqueness of the elements, thus preventing ambiguity. In APRImora, the URI is used to uniquely name the patterns and relate them to other patterns. The field Identifier is responsible for receiving the URI value, indicating the uniqueness of the patterns in APRImora through the DC2AP or DC2DP metadata, since the field Relation and its specializations are responsible for receiving URI values that associate several patterns documented in the reuse infrastructure described in this study.

In APRImora, the field Identifier receives a user-informed URI address generated using Persistent Uniform Resource Locators (PURLs)². PURLs are Web addresses that act as permanent identifiers, i.e., allow the resources to change address (server) without affecting the systems that depend upon them. Thus, the Web addresses may migrate from one domain to another without losing the reference of the allocated resource. In order to generate a URI, the user must have an administrator account on the site purl.org, fill out the fields referring to the PURL they wish to create, and await the verification of whether the URI they intend to register is available.

On the XML layer, the Namespaces allow the content syntax to be made available on the Web to be defined and its structure to be specified [13]. APRImora uses controlled vocabulary and syntax in some elements of the DC2AP and DC2DP. Using the XML layer in APRImora allows the metadata to be defined with the syntax that comprehend the DC2AP and DC2DP profiles.

The XML language allows the syntax of the content available on the Web to be defined, i.e., the content structure, the presentation form, and the content itself. However, only the syntax might not be enough. For example, based only on the term “author” of a design pattern, it cannot be inferred whether the “author” is a person or a team, nor other patterns belonging to the same “author” can be listed. In order to solve this limitation, the use of the RDF data model is proposed, presented in the third layer of the Semantic Web architecture.

These types of data are not dealt with in an RDF document, but rather their properties or information units. Hence, RDF goes beyond XML in terms of syntax and structure of the documents available on the Web, although this advance is not enough since it still does not allow the terms to be conceptualized and disambiguated [13]. In APRImora, the RDF layer is used to formalize the metadata semantics.

The ontology layer is used to describe the domain of interest. It is implemented using the OWL language, which was designed to be used by computing applications that need to process content from the information instead of only presenting it to humans [13]. This language expands the ability of inferences aided by XML, RDF, and RDFS, thus providing an additional vocabulary along with formal semantics. In APRImora, the OWL layer is used to help filling out the metadata.

SparQL is a query language used to retrieve information from an RDF document. SparQL provides protocols to query and handle RDF graphs, thus allowing several inferences [13]. This language is used perpendicularly to the RDF and OWL layers. In APRImora, the information is retrieved from repositories using the SparQL in the ARQ module belonging to the Apache Jena framework.

C. Linked Open Data

The patterns stored on the Metadata Repository are described using the Linked Data, generating a cloud among the patterns in this repository (Figure 4). The light-gray nodes indicate design patterns and the dark-gray nodes illustrate analysis patterns. This

¹ https://jena.apache.org/getting_started/index.html

² <http://purl.oclc.org/docs/index.html>

Linked Data cloud represents the connections made through the URI among the design and analysis patterns.

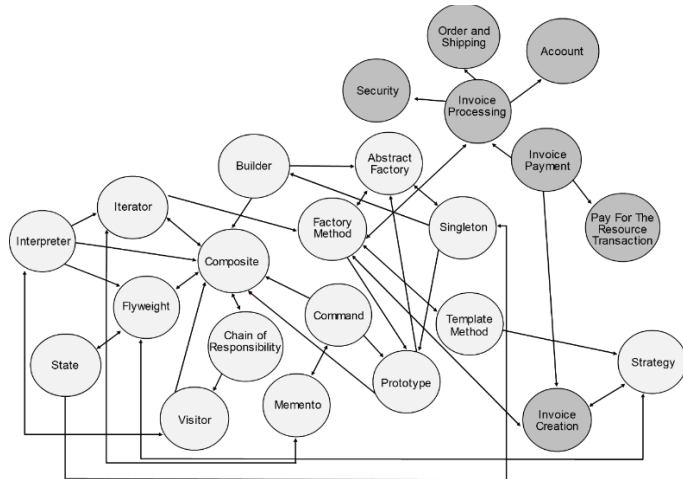


Figure 4. Linked Data cloud of part of the APRImora metadata repository.

The connections presented in Figure 4 indicate a pattern is related with others. Some patterns can be used as a set, e.g., Composite is constantly used along with Iterator or Visitor [8]. Some patterns can be used as alternatives, as is the case of Prototype used in place of Abstract Factory. Moreover, some patterns can result in similar designs, although their purposes are different, such as the structure diagrams of Composite and Decorator being similar.

The main advantage of relations among patterns is allowing the designer to have multiple ways of thinking about the patterns, thus deepening their perception on what each pattern does and in which situation it can be employed. These multiple ways allow to choose the best way to use the pattern and those that complement it, enhancing designing with patterns reuse.

Linked Open Data is a worldwide data publication effort, making them open and available to be used anywhere [23]. In order for these data to be published and connected, Linked Data must be used. Thus, it was added to the APRImora architecture through a connection with the Metadata Repository to indicate that the Linked Data cloud generated by the patterns documented and stored in this repository are now part of the Linked Open Data, which makes the patterns open and available to be used.

D. Pattern Repository

The Pattern Repository is a repository containing descriptions of diagrams expressed in XMI (XML Metadata Interchange), JPG, and PNG, which represent the solutions proposed by the patterns, thus allowing them to be used by portrayal and collaboration services. In APRImora, the Pattern Repository is made up of two repositories, Design and Analysis, which relate to each other through Linked Data. So, a design pattern can be referenced by an analysis pattern and vice-versa.

The relation between the analysis and design patterns has the advantage of sharing the experience of using these patterns, i.e., the analyst can list which design patterns can be used to implement a given analysis pattern. The same occurs regarding design patterns, in which the designer can indicate which analysis patterns that design pattern can implement.

E. Web Service and Semantic Web Service

Patterns and services can be searched for in APRImora either by a Human User through search engines or by Client Software through Web Service or Semantic Web Service.

As seen in Figure 3, the connection between the Web Service/Semantic Web Service and the Pattern Portal is two-way since the services can use the tools to carry out searches and the Pattern Portal uses these services along with its tools. For example, the metadata editors use Web Services to validate (Validation Service) the metadata fields, visualize diagrams (Portrayal Service), and download the RDF files (Access Service) that contain the pattern descriptions. The Semantic Web Service, is used by the Pattern Portal to generate an RDF document (RDF Service) and the Hyperbolic Tree.

With the search result returned, the user can use a Semantic Web Service to generate a hyperbolic tree, which is an easily understood graph on the pattern relationship. In APRImora, the central node of this tree will always be the pattern returned according to the search criteria, while the other nodes connected to the central node are patterns related with it. This is possible thanks to the use of Linked Data.

Figure 5 illustrates a hyperbolic tree corresponding to the connections of possible design patterns existing in the pattern repository, whose root node is the pattern Prototype. Using a hyperbolic tree is possible to visually navigate among the connections between the patterns.

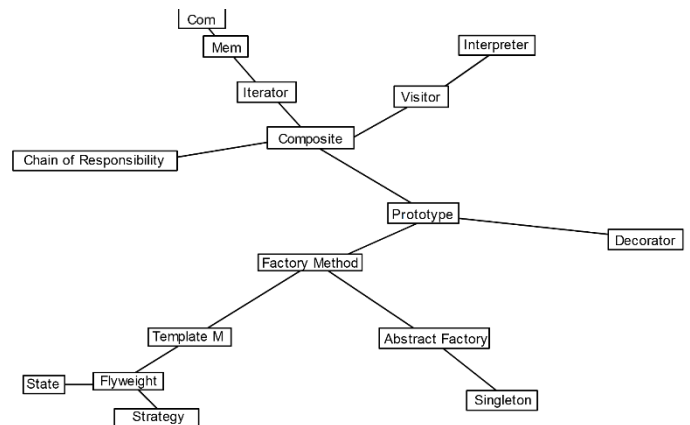


Figure 5. Hyperbolic tree with design patterns

APRI had only the component Web Service, which in the APRImora architecture is divided into two components, Web Service and Semantic Web Service. Web Service makes the following services available:

- Portrayal Service – support the visualization of the diagrams of solutions proposed by the patterns;
- Catalog Service – allows the patterns and services to be discovered and used based on their describing metadata;
- Access Service – allow accessing and downloading the patterns;
- Collaboration Service – allow designers and developers to share their use experiences to improve the patterns;

- MDA Service – allows source code to be generated from diagrams;
- Validation Service – allows metadata to be validated.

The Semantic Web Service component makes the following services available:

- RDF Service – allows RDF documents to be generated;
- Hyperbolic Tree – generates a visualization of the connections among the patterns and enables navigation.

The Web Service and Semantic Web Service components have a two-way connection with Linked Open Data, which indicates the services can carry out external searches as long as the result obtained is in a format compatible with those supported by the search carried out by APRImora. Both in APRImora and in the environment external to the infrastructure at hand, the language used to retrieve information is SparQL, while the search is done only in RDF files.

The connection between Web Service/Semantic Web Service and Linked Open Data leads to the possibility of discovering other data sources related to the patterns whenever these are published. The opposite is also possible since external agents can search the APRImora repositories using services.

V. CONCLUSIONS AND FUTURE WORK

One of the big issues faced by developers regarding the analysis and design steps is discovering reuse patterns that meet their needs. The search engines used to retrieve the patterns do not always help since a simple search returns a large number of results, many of which irrelevant to that software-design context.

The APRI was proposed to help developers search for patterns. The analysis and design patterns contained in the APRI are documented using the DC2AP and DC2DP application profiles, and later stored in their repositories. However, storing the patterns in repositories is not enough since there are hundreds of patterns and a manual search can be exhausting while search engines can return a large number of results.

To help the user better retrieve patterns contained in its repositories, the APRI was extended in this paper to an architecture based on the Semantic Web, which originates APRImora, a Pattern Reuse Semantic Infrastructure. The Semantic Web layers were used to extend the APRI since they provide means for data to be organized in a way that they can be easily interpreted by computers. Moreover, the way the Semantic Web organizes the data allows the search results to be improved. Hence, by adding Semantic Web layers to the APRI, it is expected that the information on the computing artifacts stored in the repositories are more precisely retrieved, thus allowing the user to reuse the patterns that best meet the needs.

As future work, it is intended to create a functional prototype of the APRImora so that experiments can be made to prove its efficiency. In addition, adding software agents to the APRImora is proposed to help in the automatic recovery of knowledge.

ACKNOWLEDGMENT

This work was financed by FAPEMIG, CNPq and CAPES.

REFERENCES

- [1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, n. 5. pp. 28-37, 2001.
- [2] T. Berners-Lee, "Semantic Web Concepts". 2005. [Online]. Available: <http://www.w3.org/2005/Talks/0517-boit-tbl>
- [3] T. Berners-Lee, "Linked Data". 2006. [Online]. Available: <http://www.w3.org/DesignIssues/LinkedData.html>
- [4] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data – the story so far," *Int. Journal on Semantic Web and Information Systems*, vol. 5 n. 3. pp. 01-22, 2009.
- [5] D. R. B. Cunha, B. F. Lóscio, and D. Souza, "Linked Data: da Web de Documentos para a Web de Dados," in *Livro Texto dos Minicursos ERCEMAPI*, A. M. Santana et al., SBC: Teresina, BR, 2011, pp. 79-99.
- [6] DCMI - Dublin Core Metadata Initiative, "Dublin Core Metadata Element Set, Version 1.1: Reference Description". 1999. [Online]. Available: <http://www.dublincore.org/documents/1999/07/>
- [7] M. Fowler, *Analysis Patterns: reusable object models*. Addison-Wesley Publishing, 1997.
- [8] E. Gamma, R. et al., *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Publishing, 1994.
- [9] S. Hawke, "SPARQL Query Results XML Format (Second Edition)". 2013. [Online]. Available: <http://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/>
- [10] G. Klyne, J.J. Carroll, and B. McBride, "RDF 1.1 Concepts and Abstract Syntax". 2014. [Online]. Available: <http://www.w3.org/TR/rdf11-concepts/>
- [11] N. Y. Kobashi, "Vocabulário controlado: Estrutura e Utilização". 2008. [Online]. Available: http://www2.enap.gov.br/rede_escolas/arquivos/vocabulario_controlado.pdf
- [12] D. Martin, and J. Domingue, "Semantic Web Service, Part 1," *IEEE Intelligent Systems*, vol. 22, n. 5. pp. 12-16, 2007.
- [13] F. S. Monteiro, "Web semântica e repositórios digitais educacionais na área de saúde: uma modelagem com foco no objetivo de aprendizagem para refinar resultados de busca," unpublished.
- [14] D. D. Nebert, "Developing spatial data infrastructures: the SDI cookbook". 2004. [Online]. Available: <http://www.gsdi.org/docs2004/Cookbook/cookbookV2.0.pdf>.
- [15] V. F. Pamplona, "Web Services: Construindo, disponibilizando e acessando Web Services via J2SE e J2ME". 2010. [Online]. Available: <http://javafree.uol.com.br/artigo/871485/Web-Services-Construindo-disponibilizando-e-acessando-Web-Services-via-J2SE-e-J2ME.html>
- [16] D. A. Peixoto, L. F. M. Vegi, and J. Lisboa-Filho, "DC2AP Metadata Editor: A metadata editor for an Analysis Patterns Reuse Infrastructure," *Proc. of the CAiSE'13 Forum*. pp. 138-145, 2013.
- [17] A. A. A. Ribeiro, J. Lisboa-Filho, L. F. M. Vegi, and A. P. Oliveira, "DC2DP: a Dublin Core Application Profile to Design Patterns," *Proc. of the 16th ICEIS*. pp. 209-215, 2014.
- [18] L. F. M. Vegi, J. Lisboa-Filho, G. L. S. Costa, A. P. Oliveira, and J. L. Braga, "DC2AP: A Dublin Core Application Profile to Analysis Patterns," *Proc. of the 24th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*. pp. 511-516, 2012.
- [19] L. F. M. Vegi, J. Lisboa-Filho, and J. Cromptvoets, "A machine-processable Dublin Core application profile for analysis patterns to provide linked data," *Proc. of the Int. Conf. on Dublin Core and Metadata*. pp.23-32, 2012.
- [20] L. F. M. Vegi, D. A. Peixoto, L. S. Soares, J. Lisboa-Filho, and A. P. Oliveira, "An infrastructure oriented for cataloging services and reuse of Analysis Patterns," *Proc. of BPM 2011 Workshops*, pp. 338 – 343, 2012.
- [21] L. F. M. Vegi, J. Lisboa-Filho, J. Cromptvoets, L. S. Soares, and J. L. Braga, "A Dublin Core application profile for documenting analysis patterns in a reuse infrastructure," *IJMSO*, vol.8, n. 4. pp.267-281, 2013.
- [22] W3C, "Semantic Web". [Online]. Available: <http://www.w3.org/standards/semanticweb/>
- [23] F. H. Zaidan, "LOD – Linked Open Data – Dados Abertos Vinculados". 2013. [Online]. Available: <http://itweb.com.br/blogs/lod-linked-open-data-dados-abertos-vinculados/>

Mining Universal Specification Based on Probabilistic Model

Deng Chen^{1, a}, Yanduo Zhang^{2, b}, Rongcun Wang^{3, c}, Xun Li^{2, d}, Li Peng^{4, e}, Wei Wei^{1, f}

¹ Industrial Robot Engineering Center, Wuhan Institute of Technology, Wuhan, P.R. China

² Hubei Provincial Key Laboratory of Intelligent Robot, Wuhan Institute of Technology, Wuhan, P.R. China

³ School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, P.R. China

⁴ Hubei Radio & TV University, Wuhan, P.R. China

^a chendeng8899@hust.edu.cn

^b zhangyanduo@hotmail.com

^c rcwang@hust.edu.cn

^d linuxfly@gmail.com

^e peling9901@126.com

^f weiwei@huawei-elec.com

Abstract—Class temporal specification is a kind of important program specifications, which specifies that methods of a class should be called in a particular sequence. Dynamic specification mining is a promising approach to achieve this kind of specifications automatically. However, they always infer partial specifications, that is, the mined specifications are biased to input programs or program execution traces. In this paper, we propose to mine class temporal specifications based on a probabilistic model in an online mode. Since our method can evolve mined specifications persistently, universal specifications can be achieved. To investigate our technique's feasibility and effectiveness, we implemented it in a prototype tool ISpecMiner and used the tool to perform experiments. Experimental results show that our method is promising to infer universal specifications if sufficient traces are provided for mining.

Keywords- program specification mining; Markov model; class temporal specification; dynamic analysis; program execution trace

I. INTRODUCTION

Class temporal specification (which is also referred to as component interface [1], object behavior model [2], object usage model [3], [4], etc.) is an important kind of program specifications, which imposes temporal constraints regarding the order of calls of class public methods (a public method of class c is a method that can be accessed outside c). For example, calling `peek()` on `java.util.Stack` without a preceding `push()` gives an `EmptyStackException`, and calling `next()` on `java.util.Iterator` without checking whether there is a next element with `hasNext()` can result in a `NoSuchElementException`. Client programs that violate such specifications do not obtain the desired behavior and may even crash the program [5]. However, class temporal specification is always implicit in programs and undocumented. Even when available, there is no guarantee of their consistence, completeness, and correctness. Dynamic specification mining is a promising approach to resolve the problem.

Dynamic specification mining techniques [6]-[8] run

(DOI reference number: 10.18293/SEKE2015-219)

applications with test cases generated automatically or manually, and extract specifications from program execution traces. Since dynamic specification mining techniques do not require program source code as input, compared with static specification mining [9]-[11], they can be used extensively, especially when source code is unavailable. However, existing dynamic specification miners (such as ADABU [2]) always achieve partial specifications. In order to mine specifications, these miners first run an application program and collect program execution traces into a traces file leveraging instrumentation techniques. Then, they take the trace file as input and synthesize specifications based on various kinds of sequential data mining approaches. Each run of an application will generate a trace file and corresponding specifications. The problem with this approach is that mined specifications may be biased to the application program and input traces.

In this paper, we propose to mine class temporal specifications in an online mode. Different from existing work, our approach does not save program execution traces in any file. It takes each method call from an execution trace sequentially and evolves existing specifications or creates a new one. The online approach does not require loading all traces into memory at once. Thus, it has minimum space overhead. Additionally, since execution traces extracted from different application programs can be used to refine existing specifications persistently, universal specifications may be achieved. Another characteristic of our technique is that, we describe class temporal specification using a probabilistic model extended from Markov chain. Compared with commonly used Finite State Automaton (FSA), probabilistic model has an inherent ability to tolerate noise. Above all, it can facilitate our online mining strategy.

To investigate the effect of our approach, we implemented our technique in a prototype tool ISpecMiner and used it to conduct experiments. Experimental results show that, our approach is promising to achieve universal specifications, if enough application programs are provided for learning.

The contributions of this paper are as follows:

- An online approach is used to mine class temporal specifications.
- A probabilistic model extended from Markov chain is used to describe class temporal specifications.
- A prototype tool ISpecMiner that implements our technique is presented.
- Experiments are performed to investigate the effect of our method.

The rest of this paper is organized as follows: Section II discusses related work. Section III introduces our technique. Section IV presents our experimental results. Section V gives our conclusions.

II. RELATED WORK

Generally, program specification mining techniques can be categorized into static analysis approaches and dynamic analysis approaches. These two kinds of approaches collect method call sequences (or program execution traces) in different manners. Static analysis approaches do not require running application programs. They extract method call sequences from program source code, bytecode or other artifacts based on program static analysis techniques [12]. Dynamic analysis approaches do not take program source code as input. They collect program execution traces by running instrumented application programs. After that, temporal specifications can be synthesized from method call sequences (or program execution traces) based on sequential data mining techniques.

Currently, a commonly used mining approach is based on FSA. For instance, Wasylkowski et al. [1] proposed to mine object usage models (which are finite state automata) from Java bytecode and a tool JADET was developed. Lorenzoli et al. [13] modeled class temporal specification using EFSM which extends from FSM. Alur et al. [14] synthesized FSA model of class temporal specification using L* learning algorithms combined with model checking and abstract interpretation techniques. These approaches work in a similar manner. First, they split program execution traces into a set of object usage scenarios (an object usage scenario is a method call sequence, all method calls of which have the same receiver object). Then, they reduce the problem of inferring temporal specifications from a set of method call sequences (or traces) to the well known grammar inference problem [15] by regarding method call sequences and specifications as sentences and languages respectively. As a result, a specification is described using one or multiple finite state automata, where states represent states of involved objects and transitions represent method calls. Method calls in each path from an initial state to a final state

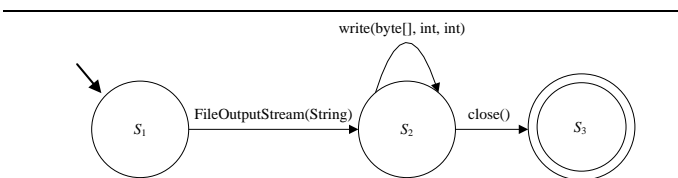


Figure 1. Temporal specification of class `FileOutputStream` described using FSA.

constitute a valid execution trace. Figure 2 shows an example of specification for class `java.io.FileOutputStream`. The specification illustrates that, to use class `FileOutputStream`, we should first initiate it through calling its constructor method. Next, we can call method `write(byte[],int,int)` multiple times to write data into the stream. Finally, method `close()` should be called to close the stream.

FSA is a kind of deterministic model with inability to tolerate noise. Ammons et al. [16] proposed to mine temporal specifications among application programming interfaces (API) or abstract data types (ADT) based on probabilistic finite state automaton (PFSA). A PFSA is a nondeterministic finite automaton (NFA), in which each edge is labeled by an abstract interaction and weighted by how often the edge is traversed while generating or accepting scenario strings. To mine temporal specifications, first an off-the-shelf PFSA learner was used to analyze scenario strings and generated a PFSA. Next, another component corer was employed to transform PFSA to NFA by discarding rarely-used edges and weights. The NFA obtained was used for program verification and manual inspection.

However, existing tools (such as Daikon [17] and ADABU [2]) always work in a two-step mode. In the first step, they collect execution traces from application programs using a tracer and then store the traces in a trace file. In the next step, they take the trace file as input and synthesize specifications. Each run of an application will generate a trace file and corresponding specifications. The problem with this approach is that results of multiple runs cannot be merged. Thus, the mined specifications are biased to the input trace file. In this work, we mine class temporal specifications based on an online approach. In addition, a probabilistic model extended from Markov chain is employed to describe specifications.

III. OUR TECHNIQUE

In this section, we present our online specification mining technique. We first provide an intuitive description of our technique and then discuss its main characteristics in detail.

A. General Approach

The working principle of our approach is illustrated in Figure 2. The tracer is responsible for collecting program execution traces from application programs via instrumentation technique. Different from existing approaches, our method does not save execution traces into any file. The tracer passes each method call of a trace sequentially to the online specification miner. The online specification miner learns class temporal specifications based on a probabilistic model. For each class, it

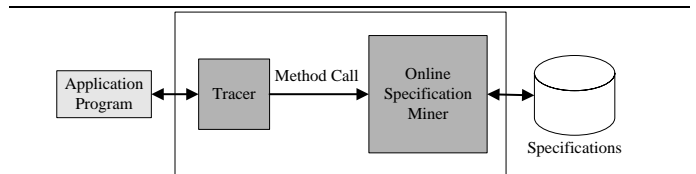


Figure 2. Working principle of our online specification mining technique.

first creates an empty specification described using the probabilistic model. Then, it evolves the probabilistic models persistently in terms of method calls passed by the `tracer`.

As we can see, our approach does not require loading all traces into memory. It refines existing probabilistic models based on a method call continuously. Therefore, compared with existing approaches, our method has lower space overhead. Furthermore, since method calls of any traces or application programs can be used to learn specifications, universal specifications may be achieved.

B. Collecting Program Execution Traces

To collect program execution traces, we should instrument application programs. Many approaches and frameworks exist to instrument Java applications statically or dynamically. We adopt Java agent technique, which is a service provided by Java since 1.5 [18]. Java agents can instrument classes at bytecode level. When a class is loaded, a Java agent catches the bytecode of this class on the fly. Then, it parses the class, injects new bytecodes. Finally, the instrumented class is returned back to the JVM.

To manipulate class bytecodes, we utilize a library `Javassist` [19], [20]. Compared with similar tools [21], [22], `Javassist` can provide the source level API, which enables programmers to edit a class file without knowledge of Java bytecodes. Furthermore, code can be inserted into class files in the form of Java source text and `Javassist` will compile it on the fly.

In order to collect program execution traces from an application program, we load a Java agent at startup using the `-javaagent` command-line switch. The agent will insert an `event writer` into the body of interested methods. Once the methods are called, the embedded `event writer` passes all necessary information regarding the method call to the specification miner for learning.

C. Mining Specification

1) Markov Chain with Final Probability

We mine class temporal specifications based on an extended Markov chain with final probability (MCF) [23]. MCF extends Markov chain by introducing a probability distribution over final states (final probability). The final probability is similar to initial probability. The difference is that final probability indicates which states a chance process should end with (rather than start from). The formal definition of MCF is given below.

DEFINITION 1 (Markov chain with final probability). A *Markov Chain with Final Probability (MCF)* M is a 4-tuple (Q, τ, π, γ) ,

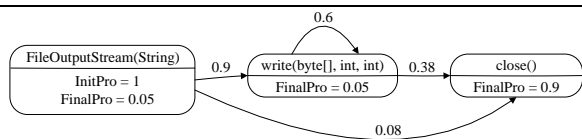


Figure 3. Class temporal specification of `FileOutputStream` described using MCF

where Q is a set of states, $\tau : Q \times Q \rightarrow [0,1]$ is the *transition probability function*, which is always described using a *transition matrix* P , $\pi : Q \rightarrow [0,1]$ is the probability distribution over *initial states*. $\gamma : Q \rightarrow [0,1]$ is the probability distribution over *final states*. The functions π and γ must satisfy the requirements: $\forall q \in Q, \sum_{q \in Q} \pi(q) = 1$ and $\sum_{q \in Q} \gamma(q) = 1$.

As shown in the definition, MCF preserves most of characteristics of Markov chain, except violation of the requirement: $\forall q \in Q, \sum_{q' \in Q} \tau(q, q') = 1$, because of introduction of final states.

Relying on MCF, we can model class temporal specification by regarding states as methods and transitions as temporal relationships among methods. Consider the class temporal specification described using FSA illustrated in Figure 1, it can be described using a MCF as shown in Figure 3. The rounded rectangles are states labeled with method signatures above the line. Arrows denote transitions with transition probability labeled beside them. `InitPro` is the probability of a state to be initial state. `FinalPro` is the probability of a state to be final state. Actually, all the states have properties `InitPro` and `FinalPro`. We omit the ones whose value is zero. From the MCF, we can see that the usage of class `FileOutputStream` should start with a method call `FileOutputStream(String)`. At the end, methods `close()`, `FileOutputStream(String)` and `write(byte[],int,int)` may be called with a probability of 0.9, 0.05 and 0.05 respectively.

2) Online Specification Learning

Our approach learns class temporal specifications described using MCF in an online mode. It accepts a method call of an OUS as input and evolves existing specifications or creates a new one.

Let R be a repository of OUSs for learning, M be the MCF specification synthesized from R , q be a state of M , t_{ij} be a transition from state i to j . Our learning strategy represents M using a weighted directed graph G_M , where nodes and edges denote states and transitions respectively. In addition, the following properties are attached to G_M .

- $ouscount(M)$ denotes the number of OUSs, which have been used to learn M .
- $emgcount(q)$ denotes the total occurrence number of state (or method) q in R .
- $initcount(q)$ denotes the count of q to be beginning method in all the OUSs of R .
- $finalcount(q)$ denotes the count of q to be end method in all the OUSs of R .
- $emgcount(t_{ij})$ denotes the total occurrence number of method pair (i, j) in all the OUSs of R .

At the beginning, we initialize G_M to be an empty graph. Then, we pick up a method call from an OUS in R sequentially

and update G_M continuously until all OUSs have been processed. For each pair of method calls q and p received currently and previously, we update G_M based on the following strategy.

- if node q does not exist in G_M , add q to G_M or else update properties associates with q .
- if edge (p, q) does not exist in G_M , add (p, q) to G_M or else update properties associated with (p, q) .
- if q is the end method of an OUS, update $ouscount(M)$.

After that, we recompute probabilities τ , π and γ according to the following equations.

$$\tau(i, j) = emgcount(t_{ij}) / emgcount(i) \quad (1)$$

$$\pi(q) = initcount(q) / ouscount(M) \quad (2)$$

$$\gamma(q) = finalcount(q) / ouscount(M) \quad (3)$$

In words, $\tau(i, j)$ is the ratio between count of transition (i, j) and that of state i in all the OUSs used for learning. $\pi(q)$ is the ratio between number of OUSs beginning with state q and the total number of OUSs. $\gamma(q)$ is the ratio between number of OUSs ending with state q and the total number of OUSs.

3) Transformation from Probabilistic Model to Deterministic Model

MCF is a kind of probabilistic model, including frequent behaviors and infrequent behaviors. In order to use the mined specifications for program verification, we should prune away infrequent behaviors (noise) in the model and obtain a deterministic model. Chen et al. [23] proposed a deterministic model Class Interface Model (CIM) and showed that it is straightforward to transform MCF to CIM.

DEFINITION 2 (Class interface model). A Class Interface Model (CIM) M of class c is a 4-tuple (M, σ, S, F) , where M is the set of public methods of c , $\sigma \subseteq M \times M$ is a binary relation on M , $S \subseteq M$ is the set of beginning methods, $F \subseteq M$ is the set of end methods. Let $p, q \in M$ be two methods, if they have the relation σ (denoted by $\sigma(p, q)$), it means that method p should be called preceding q .

A CIM of class c specifies that the usage of c should start from a method in S and then moves successively from a method m_i to m_j , where $\sigma(m_i, m_j)$, finally ends in a method of F . Any violations of the above rules are taken as errors.

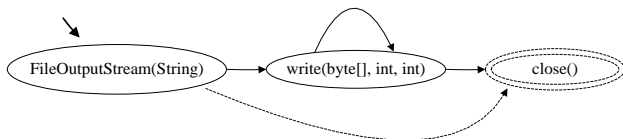


Figure 4. Class temporal specification of `FileOutputStream` described using CIM.

In order to transform MCF to CIM, we first prune away infrequent behaviors according to initial threshold (T_i), final threshold (T_f) and transition threshold (T_t), which are used to filter initial states, final states and transitions respectively. After that, we discard all the probabilities attached with states and transitions. In detail, given a MCF $\Omega : (Q, \tau, \pi, \gamma)$, we transform Ω to CIM $\Psi : (M, \sigma, S, F)$ in terms of the following rules:

- $\forall q \in Q$, add $\chi(q)$ to M , where $\chi: Q \rightarrow M$ is a function which maps a state in MCF to a method in CIM with method names the same as state labels.
- $\forall q \in Q$, if $\pi(q) \geq T_i$, add $\chi(q)$ to S .
- $\forall q \in Q$, if $\gamma(q) \geq T_f$, add $\chi(q)$ to F .
- $\forall i \in Q, j \in Q$, if $\tau(i, j) \geq T_t$, we have $\delta(i) = j$.

Figure 4 presents the CIM of class `FileOutputStream`, which is transformed from the MCF illustrated in Figure 3 based on threshold values $T_i = 0.2$, $T_f = 0.2$, $T_t = 0.2$. In the CIM, each ellipse represents a public method of the class. Arrows denote temporal relationships between pairs of methods. The methods with an arrow coming in from nowhere are beginning methods and those denoted graphically by a double ellipse are end methods. The dashed-line arrows represent the discarded transitions of MCF. As we can see, the previous MCF before transformation has three possible final states `FileOutputStream(String)`, `close()` and `write(byte[], int, int)` with a probability of 0.9, 0.05 and 0.05 respectively. The CIM discards the first and last final states because they are infrequent. In addition, the transition from state `FileOutputStream(String)` to `close()` is also pruned away due to a lower probability than T_t .

What should be noted is that results of transforming MCFs to CIMs largely depend upon values of thresholds. If thresholds are set too high, useful information will be discarded mistakenly. If thresholds are set too low, noise will remain. Even worse for our work, improper thresholds will cause unconnected CIMs. We employ the method proposed by Chen et al. [23] to compute threshold values, which can eliminate noise utmostly and obtain connected CIMs.

IV. EXPERIMENTS

In order to investigate the effectiveness of our technique, we implemented it in a prototype tool `ISpecMiner` and used the tool to mine specifications from several real-world applications. In this section, we first introduce subjects used in our experiments. Then, we present specifications mined by `ISpecMiner`.

A. Subjects

The subjects used in our experiment are listed in Table I, which consists of four real-world Java applications. We selected them based on the following criteria:

TABLE I. THE SET OF SUBJECTS

Subject	Version	Description	KLoC ^a	# Revisions	Create Date	Last Update Date
FreeMind	0.9	Mind-mapping software	22	6469	March, 2001	April, 2013
RapidMiner	5.3	Environment for machine learning and data mining	513	867	August, 2004	April, 2013
Squirrel SQL Client	3.4	Java SQL client	253	3272	June, 2004	May, 2013
OpenProj	1.4	Project management software	120	1498	January, 2008	October, 2012

a. Kilo lines of code.

- Open source software. Though ISpecMiner is a dynamic specification miner and source code is not necessary, it is helpful for us to figure out problems encountered in the mining process and validate results.
- Mature software. Mature software contains fewer bugs than the unstable one. Thus, program execution traces with less noise can be collected, which is essential for dynamic mining tools to learn precise specifications. There exist many methods to measure the maturity of software. We perform the task based on a heuristic: if an application has been maintained for a long time and undergone a large number of revisions, we believe it is mature.
- Large-scaled software. Large-scaled software can provide abundant program execution traces for learning, which is the basis of mining useful program specifications.
- Applications coming from various domains. Applications from various areas can provide diverse program execution traces, which is a strong assurance for mined specifications to be complete.

B. Mining Specifications

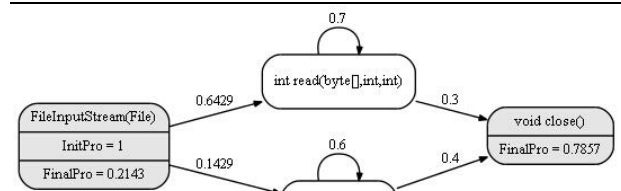
In this experiment, we used ISpecMiner to mine specifications from the subject programs presented in Table I. We ran each application once with manual input data sequentially in the order of FreeMind, RapidMiner, Squirrel SQL Client and OpenProj. After that, we examined the universality of mined specifications achieved at the end of each run. The classes that we investigated are illustrated in Table 2. We selected these classes based on the following considerations: (1) they are widely used in various Java applications and well documented; (2) they are familiar to us; and (3) since their class temporal specifications have some distinguishing characteristics (such as the usage of a class should end with a method call `close()`), we can check their validity conveniently.

Figure 5 shows an example of mined specification for class

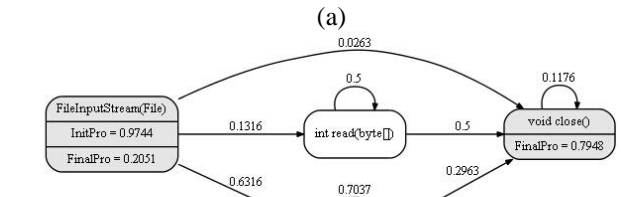
TABLE 2 INVESTIGATED CLASSES

	Class		Class
1	java.io.FileInputStream	6	java.io.InputStreamReader
2	java.io.BufferedReader	7	java.io.PushbackInputStream
3	java.io.FileOutputStream	8	java.io.FileReader
4	java.io.ByteArrayOutputStream	9	java.io.PrintWriter
5	java.io.BufferedWriter	10	java.util.Stack

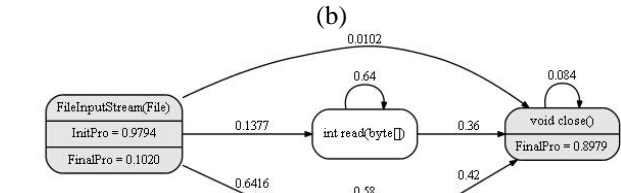
java.io.FileInputStream, where (a), (b), (c) and (d) were achieved when we finished the run of subject programs FreeMind, RapidMiner, Squirrel SQL Client and OpenProj respectively. As we can see, along with more applications used for mining, the specification grew universal, that is, more states and transitions were added to the specifications. For example, after the run of application



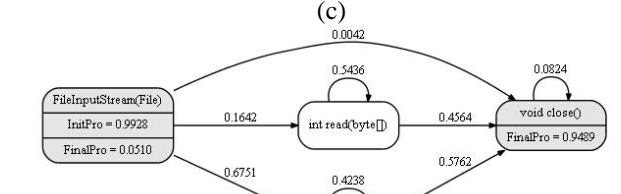
Markov Model of class java.io.FileInputStream (14, Sun Nov 02 15:56:50 CST 2014)



Markov Model of class java.io.FileInputStream (39, Sun Nov 02 16:05:27 CST 2014)



Markov Model of class java.io.FileInputStream (124, Sun Nov 02 16:10:27 CST 2014)



Markov Model of class java.io.FileInputStream (296, Sun Nov 02 16:36:27 CST 2014)

Figure 5. Example of mined probabilistic specification

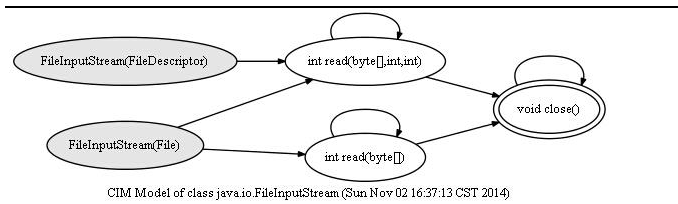


Figure 6. Example of mined deterministic specification

RapidMiner, a new state `FileInputStream(FileDescriptor)` and transition `<FileInputStream(File), close()>` shown in Figure 5 (b) were added to the previous specification illustrated in Figure 5 (a). Furthermore, since more applications were used to evolve the specification, probabilities of normal and abnormal behaviors (such as the `FinalPro` of state `close()` and that of state `FileInputStream(File)`) in the specification were increased and decreased respectively. Finally, the gap between probabilities of useful information and noise will become large, and then correct deterministic specifications can be achieved by transforming the final MCF to CIM. The specification of class `FileInputStream` described using CIM is illustrated in Figure 6, which is transformed from the final MCF under threshold values computed according to the method by [23]. After a close investigation, the CIM is correct and consistent with JDK documentations.

In conclusion, we used `ISpecMiner` to mine class temporal specifications from four real-world Java applications and examined specifications of 10 JDK classes. We found that our technique can refine mined specifications persistently. In addition, the probabilities of useful information will be enhanced, which is beneficial for transforming probabilistic models to correct deterministic models. `ISpecMiner` and other specifications mined in our experiment can be obtained at the URL <http://ispecminer.com>.

V. CONCLUSIONS

In this paper, we proposed an online program specification mining approach based on an extended Markov model. Different from existing approaches which work in a two-step mode, our method does not require saving collected program execution traces into a trace file. It first creates an empty probabilistic model for each class, and then evolves the probabilistic model persistently based on method calls in input traces. Since our approach does not require loading traces into memory at once, it has low space overhead. Additionally, if enough applications are provided for mining, universal specifications may be achieved.

ACKNOWLEDGMENT

Supported by Natural Science Foundation of Hubei Province (No. 2014CFB1006).

REFERENCES

[1] A. Wasylkowski, A. Zeller, C. Lindig, Detecting object usage anomalies. Proceedings of the 6th Joint Meeting of the European Software

Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM, Dubrovnik, 2007.

[2] V. Dallmeier, C. Lindig, et al., Mining object behavior with ADABU. Proceedings of the 2006 International Workshop on Dynamic Systems Analysis, ACM, Shanghai, 2006.

[3] M. Pradel and T.R. Gross, Automatic generation of object usage specifications from large method traces. Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, 2009.

[4] A. Wasylkowski, Mining object usage models. Companion to the Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, 2007.

[5] M. Pradel and T.R. Gross, Leveraging test generation and specification mining for automated bug detection without false positives. Proceedings of the 34th International Conference on Software Engineering, Zurich, Switzerland, 2012, 288-298.

[6] M.D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering, vol. 27, 2001, 99-123.

[7] M. Gabel and Z. Su, Javert: fully automatic mining of general temporal properties from dynamic traces. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, Atlanta, 2008.

[8] J.H. Perkins and M.D. Ernst, Efficient incremental algorithms for dynamic detection of likely invariants. SIGSOFT Softw. Eng. Notes, vol. 29, 2004, 23-32.

[9] M.K. Ramanathan, A. Grama, and S. Jagannathan, Static specification inference using predicate mining. SIGPLAN Not., vol. 42, 2007, 123-134.

[10] S. Thummalapenta, and T. Xie, Alattin: mining alternative patterns for defect detection. Automated Software Engineering, vol. 18, pp. 293-323, 2011.

[11] D. Lo, G. Ramalingam, et al., Mining quantified temporal rules: formalism, algorithms, and evaluation. Science of Computer Programming, vol. 77, pp. 743-759, 2012.

[12] D. Chen, R. Huang, et al., Improving static analysis performance using rule-filtering technique. Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering, 2014.

[13] D. Lorenzoli, L. Mariani and M. Pezz, Automatic generation of software behavioral models. Proceedings of the 30th International Conference on Software Engineering, ACM, Leipzig, 2008.

[14] R. Alur, P. Cerny, et al., Synthesis of Interface Specifications for Java Classes. SIGPLAN Not., vol. 40, 2005, 98-109.

[15] J.E. Cook and A.L. Wolf, Discovering models of software processes from event-based data. ACM Trans. Softw. Eng. Methodol., 7(3), 1998, 215-249.

[16] G. Ammons, R. Bodik and J.R. Larus, Mining specifications. SIGPLAN Not., vol. 37, 2002, 4-16.

[17] M.D. Ernst, J.H. Perkins, et al., The Daikon system for dynamic detection of likely invariants. Science of Computer Programming, vol. 69, 2007, 35-45.

[18] P. Caserta and O. Zendra, JBInsTrace: a tracer of Java and JRE classes at basic-block granularity by dynamically instrumenting bytecode. Science of Computer Programming, vol. 79, pp. 116-125, 2014.

[19] S. Chiba and M. Nishizawa, An easy-to-use toolkit for efficient Java bytecode translators. Proceedings of the 2nd International Conference on Generative Programming and Component Engineering, Springer-Verlag, New York, 2003.

[20] M. Tatsubori, T. Sasaki, et al., A bytecode translator for distributed execution of "legacy" Java software. Proceedings of the 15th European Conference on Object-Oriented Programming, Springer-Verlag, 2001.

[21] ASM, <http://asm.ow2.org>.

[22] BCEL, <http://commons.apache.org/proper/commons-beel>.

[23] D. Chen, R. Huang, et al., Mining class temporal specification dynamically based on extended Markov model. International Journal of Software Engineering and Knowledge Engineering, 2014, in press.

Topic Matching Based Change Impact Analysis from Feature on User Interface of Mobile Apps

Qiwen Zou¹, Xiangping Chen^{2,*}, Yuan Huang^{1,3}

¹School of Information Science and Technology, Sun Yat-sen University, Guangzhou, China, 510006

²Institute of Advanced Technology, Sun Yat-sen University, Guangzhou, China, 510006

³Ocean University of China, Qingdao Haier Intelligent Home Appliance Technology Co.,Ltd, Qingdao, China

Email: cathyqzq@163.com, chenxp8@mail.sysu.edu.cn, huangyjn@gmail.com

Abstract—The complexity of mobile applications often lies in the user interface (UI). To update function provided by UI or just fix bugs related to UI, software maintainers primarily need to obtain the location of source code implementation and detect change set. Since UI related feature is tightly related to the class containing the declaration of the UI component, this paper proposes a topic matching based change impact analysis method from feature on user interface of mobile apps. Our approach combines LDA model with program dependency to realize the change impact analysis. Considering app's small scale and few comments, a novel preprocessing method combining *tf-idf* with term weight based on structural information is applied to LDA model. Experiments on 16 update records of 4 open source apps show the effectiveness of our proposed method.

I. INTRODUCTION

With the rapid development of mobile application industry, most mobile applications (i.e., apps) are updating frequently, which challenges software maintainers. During software maintenance, developers must spend much efforts on program comprehension when related documents are missing and even original developers are no longer available. However, to seek out relative classes manually is difficult and time-consuming. Change impact analysis [12] is always a special topic of determining potential consequences of a proposed change.

Different from traditional software, the user-friendly of apps has become a key point to attract users thus modifying user interface (UI) is frequent. In addition, app has to deal with users' various requirements through UI and this can be error prone for UI with the complexity of the demands [1]. Thus we guess frequent function update is associated with app UI and it's an active area of software maintenance tasks. To validate our conjecture, we manually browse 1007 update records of 306 apps collected from Google Play Store¹, in which 458 changes are related to function provided by UIs, excluding these changes such as data storage, configuration file and ambiguous changes (fix bugs, improve performance, etc.). The rate which reaches 45% shows a frequent modification of source code related to UI. Detailed data are available in the online appendix². Further, we investigate keywords on UI and want to know how many words appearing on UI will exist in

the topics. Results suggest an average of 35%, which indicates that function provided by UI can be well described by topics.

To update function provided by UI or just fix bugs related to UI, software maintainers primarily need to obtain the location of source code implementation and detect change set (i.e., classes that may be modified to accomplish an update of app UI). Feature location is always an option and then change impact analysis can help to find relative classes. A UI related software feature is tightly related to the class containing the declaration of the UI component providing its corresponding function. In this context, locating the feature on UI can automatically detect an initial class for impact analysis.

In recent years, text retrieval model such as Latent Dirichlet Allocation (LDA) [3] is generally used to locate feature while impact analysis depends on program dependency [9, 10, 12]. However, to do impact analysis for the update of feature provided by UI, those information implied by keywords on UI is very important, using LDA to mine relative classes and combining program dependency, we expect to obtain improved recommended list of classes. Considering app's small scale and few comments, using entire source code corpus not just identifiers and comments is essential. If so, the words extracted may contain too much noise, extracting topic directly from source code may be less effective. Novel preprocessing techniques term frequency-inverse document frequency (*tf-idf*) and structural information based term weight can be applied to filter out less meaningful words and make topics prominent.

In this paper, we propose a topic matching based change impact analysis method for maintaining UI of apps. Our approach starts from locating software feature on UI to its implementation in source code as the initial class for impact analysis. Then, we combine LDA with program dependency to realize change impact analysis. For LDA, a novel preprocessing method combining *tf-idf* with term weight is applied.

We have conducted experiments on 16 update records of 4 open source apps to evaluate the effectiveness of our method, results show that our approach works well for recommending appropriate classes for corresponding feature on UI.

II. RELATED WORK

A. Feature Location

Feature location, also called concept location, is a program comprehension phase during software maintenance to detect

¹<http://www.androidcentral.com/google-play-store>

²<http://research.defool.me/dataset/website/1.html>

DOI reference number: 10.18293/SEKE2015-078

source code implementation of features of target system [2]. In recent years, most researches on feature location have focused on (semi-)automated techniques to alleviate manual operation. The most common analysis techniques are static, dynamic, textual, or a blend of several analysis approaches. Static analysis just uses source code text. Independently and in parallel, some other researches [6, 7] use dynamic information (i.e., execution trace) gathered from scenarios to locate features.

In particular, textual analysis based on modern information retrieval technique, LDA [3] and LSI [4], has been increasing popular. Marcus et al. [4] propose LSI-based feature location. Dynamic analysis combining with LSI (SITIR) results in a better performance comparing with LSI alone [7]. Lukins et al. [8] have evaluated LDA-based feature location. Experiments with Eclipse and Mozilla suggest LDA-based approach is more effective than using LSI for this task. An approach based on genetic algorithm is proposed by Annibale Panichella [10] to detect a near-optimal configuration for LDA which leads to a higher accuracy of feature location. Considering the structural characteristics of source code different from natural language, Blake Bassett [9] introduces a novel term weighting scheme for LDA to improve accuracy of feature location.

In our research of detecting relative classes for corresponding function provided by UI, feature location with characteristics of apps is used to find initial class for change impact analysis. Meanwhile, considering the effectiveness of LDA for detecting functional related classes in those previous works, we combine the information mined by LDA with program dependency to perform impact analysis.

B. Change Impact Analysis

Change impact analysis is used to determine the potential consequences of a proposed change during software maintenance [12]. In recent years, change impact analysis for software maintenance is hot. Acharya, M. et al. [11] design a static program slicing based method to do change impact analysis for large and evolving industrial software systems. Malcom Gethers et al. [13] configure a best-fit including information retrieval, dynamic analysis, and data mining of past source code commits to present an adaptive approach to perform impact analysis from a proposed change. Hoa Khanh Dam Dam [12] makes efforts on impact analysis for client-based systems. On the whole, there have various techniques supporting change impact analysis from procedural to object-oriented system [11, 12, 13, 14, 15].

Different from those researches, our method starts change impact analysis with a recommended class which is auto-located with our tool while traditional approach selects an initial class that needs to be changed by developers. In addition, we perform change impact analysis by combining linguistic information with structural dependency of source code.

III. APPROACH

A. Approach Overview

Fig. 1 shows the overview of our approach. In the general framework, users provide interface feature as a query and a

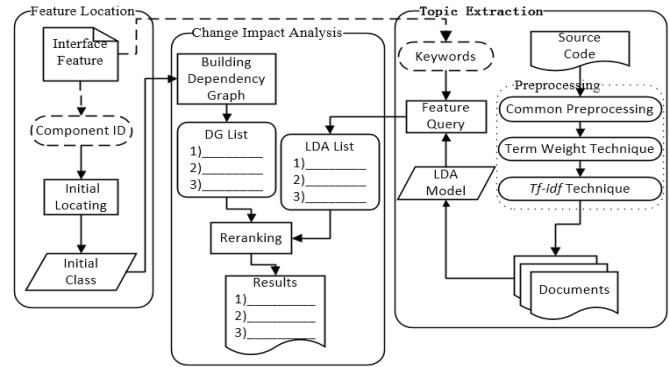


Fig. 1. The approach overview

recommended list related to the function of the UI is identified with which developers can easily detect relative classes.

Our approach starts from locating software feature on UI of mobile apps to its implementation class in source code by matching the ID of UI component.

For topic extraction, novel preprocessing techniques, *tf-idf* and term weight, are applied. Term weight technique based on the conjecture that the importance of words among different entities (e.g., classes, methods, attributes and others) are different and *tf-idf* technique stemming from information retrieval are applied to filter out less meaningful words and core words are taken as input for LDA.

Two class lists ranked based on dependency relation and topic matching degree are taken as input to rank its possibility of being impacted when maintaining the app feature on UI.

B. Feature Location

For change impact analysis, the primary step is locating the initial source code implementation of function provided by app UI, feature location is always an option.

Considering the special characteristics of apps, we model feature on UI of apps as $feature = \langle ID, keywords \rangle$, in which ID is the unique identifier of UI component in the whole app project and keywords are core words on UI or close related to this component (i.e., words existing in the ID). When text information on UI is too long and indistinctive or there has no words on the corresponding component, words in ID are considered to build keywords. Keywords are used to construct query for LDA model described in next part. ID is used to locate the initial class declaring this component. Based on our previous work on searching UI component of apps [18], we develop an auto-locating tool³ to find ID and detect the initial class by just clicking this component using a screenshot of the UI. This tool is developed based on the characteristics of apps that every component has unique ID and unique location on its belonging UI. When user is running an app and he wants to modify one component or function provided by the component on a UI, our tool can help to locate the initial location.

Fig. 2 represents examples of optimization of *message list* in project *Faceless* and *music scan* in project *Kjmusic*. The red rectangular box in Fig.2 (a) shows a message list in which

³<http://research.defool.me/uidroid/>

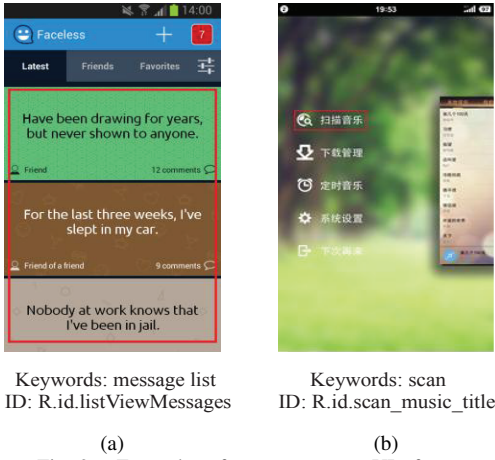


Fig. 2. Examples of components on UI of apps.

text is too long and words are less significant, we use core words in the id *R.id.listViewMessages* to build keywords. If we need to alter the display of message or add information on the list, we can click this part and locate the initial class. And in Fig.2 (b), we build keywords with text information (scan music) in the red rectangular box which is distinctive after being translated into English. We can click this part to obtain corresponding initial class and update the function of music scan.

C. Topic Extraction

The generation of topic model has following steps. Abstract Syntax Tree (AST⁴) can be used to extract the information of source code. Then, the documents are preprocessed. LDA outputs the word-topic probability distribution and the topic-document probability distribution. Thus, LDA model is constructed and can be queried with keywords of feature on UI.

1) *Source Code Preprocessing*: Common preprocessing steps include identifiers splitting, abbreviations expanding, removing stop words and stemming. In our method, we use novel preprocessing techniques, term weight and *tf-idf* to process source code corpus aiming at filtering out noise words and making topics prominent.

a) *Term weight technique*: This technique is proposed based on the experience that term in different entities (i.e. class, method etc.) has different importance by Girish Maskeri [16]. Considering the hidden but important information, term weight is taken into account to make important terms outstanding.

Empirically, a weight-based rule $f : Ttype \xrightarrow{\text{yields}} v$ that assigns various positive integers v to five types of terms (i.e., all types $T = \{\text{term in class names, term in method names, term in attribute names, term in comments, term in others}\}$) is applied. To differentiate the importance, for example, $v(\text{class})$ is assigned higher than $v(\text{method})$ because in object-oriented software system, class as functional implementation of domain problem, it is more promising to acquire the intended functional knowledge encoded in the class name than method. And empirically, other values $v(\text{comment})$, $v(\text{attribute})$, $v(\text{other})$ are assigned diminishingly. Then, we use formula

⁴c2.com/cgi/wiki?AbstractSyntaxTree

$$weight_{i,j} = \sum_{Ttype \in T} v(Ttype) \times n_{i,j} \quad (1)$$

to calculate weight sum of term i in document j . $n_{i,j}$ denotes the number of occurrences of i in the forms of $Ttype$ in document j . Further, we normalize term weight $weight_{i,j}$ to $\omega_{i,j}$ with the formula

$$\omega_{i,j} = \frac{weight_{i,j}}{\sum_{k \in D} weight_{k,j}} \quad (2)$$

because different documents have different size of vocabulary and $\omega_{i,j}$ reflects term's importance to the document j .

b) *tf-idf technique*: This technique is used to evaluate the importance of a word to a document in corpus and has been widely used in the domain of information retrieval. In our study aiming at removing noise words, *tf-idf* is applied.

tw (term weight) denotes the proportion of word i in document j . $tf_{i,j}$ is the number of occurrences of word i in document j and m stands for the number of different words occurring in document j .

$$tw_{i,j} = \frac{tf_{i,j}}{\sum_{k=1}^m tf_{k,j}} \quad (3)$$

tf-idf (term frequency-inverse document frequency) is proposed with the principle that the importance of word i to document j is in proportion to $tf_{i,j}$ while inversely proportional to the number of documents df_i containing the word i . And n is the number of all documents in the corpus.

$$tf - idf_{i,j} = tw_{i,j} \times \log \frac{n}{df_i} \quad (4)$$

Both in term weight and *tf-idf* technique, threshold needs to be set to filter out less meaningful words. A word with higher value is more representative of the document than others. We use cut points δ_{weight} and δ_{tf-idf} . Word i in the document j will be retained if $\omega_{i,j} > \delta_{weight}$ and $tf - idf_{i,j} > \delta_{tf-idf}$, if the weight $\omega_{i,j}$ or $tf - idf_{i,j}$ for a word is low, that means the word is not significant and can be considered as noise word.

2) *Model Generation*: Latent Dirichlet Allocation (LDA) is a probabilistic generation model from the term occurrences in a corpus, proposed by Blei et al [3]. Source code documents are taken as input for LDA, the documents are considered unstructured and described as bag-of-words in which the order of the words is neglected. Through training, LDA expects to obtain two matrix $\theta_d = \langle p_{t1}, p_{t2}, \dots, p_{tk} \rangle$ and $\varphi_t = \langle p_{w1}, p_{w2}, \dots, p_{wn} \rangle$. For each document d , p_{ti} denotes probability that d maps to topic t_i . For each topic t , p_{wi} denotes probability that word w_i belongs to t . For those results, documents having the same relevant topic are grouped into the same cluster.

In our approach, the classes of app source code are taken as a collection of documents. For that, we use term weight technique and *tf-idf* technique for preprocessing, appropriate thresholds should be set to choose the most likely words reflecting corresponding document, which lays a solid foundation for our purpose of change impact analysis. Therefore,

we invite three graduate students of Sun Sat-sen University to manually check the words retained by preprocessing with a large number of experiments through reduplicated adjusting and feedback, so that near-optimal thresholds can be obtained. To avoid any bias, students are not aware of the experimental goals.

We apply LDA to this entire collection of preprocessed data with Gibbs sampling. The number of Gibbs iterations n is required while every iteration samples a topic for each word. In addition, the Dirichlet hyperparameter for topic proportions ϑ and the Dirichlet hyperparameter for topic multinomials β need to be set to control the smoothing of the model. For those configuration parameters, we use suggested values from [17], $\vartheta = 0.5$, $\beta = 0.1$. Moreover, we set the number of topics and the number of top words in a topic by taking account of both app scale and the number of keywords on app UI.

3) *Feature Query*: As is illustrated in the section of introduction, we find that core words on UI will exist in topics. In that case, feature on UI can be matched with a topic and even mapped to source code classes. Having modeled feature on UI, we use keywords of feature as query, those words have been processed so that they can be matched with topic words. For each query, we compute the similarity of the feature and all topics and select the most similar topic using words matching (i.e., the topic is more relative when more words exist in both query and this topic). Then topic-document distribution is used to sort classes with descending order, we name it as LDA list.

D. Change Impact analysis

Having located the initial class, change impact analysis is used to detect recommended list. Dependency graph (actually a tree, the child B of a node A is its relative class, including two cases that A depends on B and B depends on A) starting with the initial class can be obtained. This dependency graph gives relative classes and dependency depth and can be constructed as a dependency list (DG list) in which the initial class is always in the first place, and the children of a node have no certain sequence. Source code class with smaller depth indicates it is more likely to be changed to adapt to the update. To keep the parent-child relationship, parent class is attached to every class in the DG list.

Finally, an improved recommended list will be generated by change impact analysis combining LDA list with DG list. Topics are functional description of source code, and dependency graph represents structural relations. However, in DG list, a class as a child node may have higher probability assigned by LDA than a class as parent node when parent class is used as an interface to this child class and doesn't implement core function. In theory, the parent class is important for feature update. In that case, we reassign the probability P of every class A in the LDA list considering the probabilities of its all children B_1, B_2, \dots, B_n , namely,

$$P(A) = \max(P(A), P(B_1), P(B_2), \dots, P(B_n)) \quad (5)$$

where n is the number of A 's all children. And the parent-child relationship can be detected in the DG list. In addition, the class in LDA list may be unreachable in DG list, in most

TABLE I
THE PERCENTAGE OF KEYWORDS AND TOPIC WORDS

App project	Classes	Words on UI	Keywords	$N_t = N_i$	$N_t = 2N_i$	$N_t = 3N_i$
Lightning-Browser	21	131	74	24.32%	39.19%	39.19%
Notify	95	339	114	26.32%	35.09%	35.09%
Jamendo	115	70	52	21.15%	28.85%	28.85%
EasyToken	24	198	78	15.38%	24.36%	25.64%

cases, this class is not related to the function of the UI and can't be impacted by the update, so we remove it from LDA list. Consequently, the order of classes in the topic is optimized and we obtain final recommended list.

IV. EXPERIMENTS

We have conducted an empirical study to evaluate the effects of our method. In this section, we discuss the significance of change impact analysis from feature on app UI, as well as present and evaluate the resulting data.

A. Research Motivation

The key point of our research is based on the following question:

Is the change impact analysis from feature on app UI meaningful and promising?

We extract words on app UI, the data are simply preprocessed and we remove repeated words. Meanwhile, we use LDA to extract topics, composed of a couple of words, from source code corpus. We expect to validate that the function of UI can be well described by topics. Representative results are shown in Table I. The percentage of how many words existing on UI are in topics lists in the five column when the number of topic words N_t equals the number of keywords N_i , and when N_t is double of N_i the percentage lists in the six column, and so on.

From Table I, we can see that core words on UI will appear in topics, the large percentage is near 40% when the rate is double and it is lower when $N_t = N_i$. However, many experiments show no larger percentage when we continue increasing the number of the topic words. In that case, we go deep in the projects to check those unmatched words, such as *please, sure, thanks* for *EasyToken*. Obviously, those words are less significant for this app comparing with these words such as *easy token*. Therefore, we can conclude that those unmatched words are not core words, if appropriate number of topics and number of top words in a topic are set, the query using keywords on UI can find matched topic, and then find corresponding classes with different probability. For more data, this online appendix⁵ is available.

B. Data Set & Effectiveness Measure

Our approach is used to recommend a list of classes which are probably affected by maintaining a software feature on UI. We evaluate our method with update history of apps to see the position of changed classes related to an update of UI in the list. As a result, we choose open source apps with well-written source code and available update records.

⁵<http://research.defool.me/dataset/website/2.html>

TABLE II
THE EXPERIMENT DATA

App project	Change	Update Date	All Classes	Update Classes	Change Description	Feature-keywords
Oschina	01	2014-02-19	162	9	Fixes the function of report message	report message
	02	2014-02-24	162	7	Added welcome screen to start different figure with different festivals	start welcome
	03	2014-02-10	162	1	Fixes flashing with tweet audio player	audio player tweet
	04	2014-03-03	162	5	Fixes the keyboard up when refreshing detail message	detail editor
	05	2012-09-14	162	2	Fixes a bug that users cant use the camera to upload new image	user image editor
Kjmusic	06	2014-01-29	41	5	Optimization of scan interface	scan
	07	2014-01-28	41	1	Repair the bug that current play pictures is hidden after disappearing	player picture
	08	2014-01-28	41	2	Optimization of lyrics playlist UI display interface	lrc
	09	2014-01-15	41	4	Repair logic error of playing a looping pattern	loop mode play
Faceless	10	2014-12-09	30	5	Display approximate distance to message author on Android	message list
	11	2014-12-09	30	2	Added location input when composing message in Android	location input
	12	2014-12-11	30	5	Added 'Nearby' messages feature on Android	nearby message
	13	2014-12-15	30	1	Hide secondary options in message compose window by default	advance option expand message
Jamendo	14	2012-09-13	114	1	Fixes preset naming	preset equalizer
	15	2012-07-05	114	2	User can customizer equalization	customizer equalization
	16	2011-04-14	103	9	Added paginated retrieval of all Album's tracks	album track

We choose 4 open source apps in our experiment: Oschina⁶, Kjmusic⁷, Faceless⁸, Jamendo⁹. Oschina is an open source china community for sharing open source software. Faceless is an anonymous social software where you can talk freely. Kjmusic and Jamendo are two music players. The words on UI of Oschina and Kjmusic are Chinese, we use a translator to translate them into English during feature location.

Table II summarizes app information that we use to conduct the experiments, including update date, the number of classes and update classes, change description and feature-keywords. We use a number to denote a change in our paper.

To evaluate the performance of our method, we use the effectiveness measure in [9] to evaluate our results. Descriptive statistics is to go deep in the ranked list to check the position of the updated classes including min, median, max. And min, median, max represents the rank of first class, middle class, last class that are related to the update, respectively.

In addition, we change mean reciprocal rank (MRR) as the average of the reciprocal of the location of relevant classes:

$$MRR = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{1}{r_i} \quad (6)$$

where C are all classes related to an update, and r_i is the rank of the relevant class in the recommended list. A higher MRR implies a better results.

C. Results and Analysis

In this section we represent the results of topic matching based change impact analysis method from feature on UI for four apps. For the reason that it's firstly proposed, we use traditional LDA results without considering term weight and $tf-idf$ (LDA), LDA list (CLDA) and DG list (DG) to compare with and discuss the advantages of our method.

⁶<http://git.oschina.net/oschina/android-app>

⁷<http://git.oschina.net/kymjs/KJmusic>

⁸<https://github.com/delight-im/Faceless>

⁹<https://www.jamendo.com/en/>

1) *MRR*: Fig. 3 shows MRR for 16 changes described in Table II. Compared with LDA, CLDA and DG, our method generally obtains higher MRR than others which implies a better performance. And obviously, LDA with novel preprocessing obtains higher accuracy than traditional LDA. Sometimes, our method seems poorer than DG when doing minor changes, actually they are the same because the probabilities of the first few classes are the same such as change 15, the probability of the first class is the same as that of the second class. In addition, it's obvious that there have break points in the DG line for change 03, 07, 13, 14 because we don't show MRR of DG when there is only one changed class. For that, feature location can always detect the initial class and the comparison is less promising.

2) *Descriptive Statistics*: This part we report the descriptive statistics for 16 changes of our method compared with LDA, CLDA and DG in Table III.

The character “-” in the table denotes meaningless results as explained in MRR. We note a better performance (i.e., lower rank) of our method compared with LDA, CLDA and DG. In addition, the results of traditional LDA show that topics are scattered and its accuracy is lower than CLDA. Our method takes advantage of both DG and CLDA so that min is almost 1 (some are not because the probabilities of the first

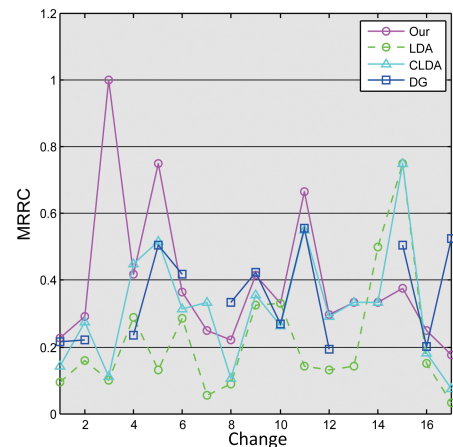


Fig. 3. MRR of our method compared with LDA, CLDA and DG

TABLE III
DESCRIPTIVE STATISTICS OF OUR METHOD COMPARED WITH LDA, CLDA, DG

Change	Our Method			LDA			CLDA			DG		
	Min	Median	Max	Min	Median	Max	Min	Median	Max	Min	Median	Max
01	1	7	105	2	76	162	2	106	161	1	22	94
02	1	5	70	1	106	152	1	16	162	1	13	60
03	1	1	1	10	10	10	9	9	9	-	-	-
04	1	4	7	2	4	6	1	3	6	1	23	25
05	1	1.5	2	4	46	84	1	15	30	1	37.5	75
06	1	8	11	1	9	39	1	6	40	1	3	18
07	4	4	4	18	18	18	3	3	3	-	-	-
08	3	6	9	8	14	20	7	10.5	14	2	4	6
09	1	5	30	1	20	40	1	9	39	1	3.5	9
10	1	5	17	1	4	29	1	24	30	1	12	19
11	1	2	3	4	16.5	29	1	5.5	10	1	5	9
12	1	7	15	3	14	29	1	11	26	2	9	19
13	3	3	3	7	7	7	3	3	3	-	-	-
14	3	3	3	2	2	2	3	3	3	-	-	-
15	2	3	4	1	1.5	2	1	1.5	2	1	40	79
16	1	9	90	1	31	115	1	36	92	1	13	24

several classes are the same) and most of the classes that are updated are in lower rank. However, this combination may make some results (larger rank) worse when CLDA and DG interact together. For change 02, our method obtains lower rank except max because the max of CLDA is large that affects our recommended list. For that, Fig. 3 shows overall results that the effect is little and our method is effective for change impact analysis from feature on app UI.

V. CONCLUSION AND FUTURE WORK

In this paper we propose topic matching based change impact analysis from feature on UI of mobile apps. Focusing on the function of app provided by UI, we firstly model it with keywords and ID and help users find appropriate classes related to the update of the function.

In our method, we develop a tool to locate the initial location by just clicking this component using a screenshot of the UI. Then, we use LDA to model source code in which novel preprocessing techniques (i.e., term weight, *tf-idf*) are applied and keywords related to UI are used as query to acquire proper class list with descending probability. Finally, dependency graph (DG) starting with initial class is detected, we take the advantage of DG and LDA with novel preprocessing techniques to do impact analysis, experiments with four apps show a better performance of our method.

In future work we plan to investigate change impact analysis from feature on UI at method level. In addition, we find some updates related to UI just modify the layout file (i.e., .xml), how to discover those files is also meaningful.

ACKNOWLEDGMENT

This research is supported by NSFC-Guangdong Joint Fund (No. U1201252), the Educational Commission of Guangdong Province (No. 2013CXZDB001), the Fundamental Research Funds for the Central Universities, and the National Science & Technology Pillar Program (No. 2012BAH12F02).

REFERENCES

[1] Zhifang Liu, Xiaopeng Gao and Xiang Long, *Adaptive random testing of mobile application*, in Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCT 10), IEEE Computer Society, Washington, DC, USA, 2, 297-301.

[2] B. Dit, M. Revelle, M. Gethers, and D. Poshvanyk, *Feature location in source code: a taxonomy and survey*, Journal of Software: Evolution and Process, vol. 25, pp. 53-95, 2013.

[3] Blei, D. M., Ng, A. Y., Jordan M I, and Jordan, M. I. , *Latent Dirichlet Allocation*, Journal of Machine Learning Research, vol. 3, pp. 993-1022, 2003.

[4] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R., *Indexing by Latent Semantic Analysis*, Journal of the American Society for Information Science, vol. 41, pp. 391-407,1990.

[5] A. Marcus, A. Sergeev, V. Rajlich, and J. Maletic, *An information retrieval approach to concept location in source code*, in Proc. of the 11th Working Conf. on Reverse Engineering, 2004, pp. 214C223.

[6] Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L., and Koschke, R., *A systematic survey of program comprehension through dynamic analysis*, Software Engineering, IEEE Transactions on, 2009, 35(5): 684-702.

[7] D. Poshvanyk, Y. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, *Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval*, IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 420C432, Jun. 2007.

[8] S. Lukins, N. Kraft, and L. Etzkorn, *Source code retrieval for bug localization using latent Dirichlet allocation*, in Proc. of the 15th Working Conf. on Reverse Engineering, 2008.

[9] B. Bassett and N. A. Kraft, *Structural information based term weighting in text retrieval for feature location*, in IEEE Int'l. Conf. on Program Comprehension, 2013, pp. 133-141.

[10] Panichella A, Dit B, Oliveto R, et al., *How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms*, in ICSE, 2013, pp. 522C531.

[11] Acharya, M., Robinson, B, *Practical change impact analysis based on static program slicing for industrial software systems*, in Proceedings of the 33rd international conference on software engineering. ACM, 2011.

[12] Hoa Khanh Dam Dam, *Automated change impact analysis for agent systems*, in Software Maintenance (ICSM), 2011 27th IEEE International Conference on (pp. 33-42). IEEE.

[13] M. Gethers, H. H. Kagdi, B. Dit, and D. Poshvanyk, *An adaptive approach to impact analysis from change requests to source code*, in ASE, 2011, pp. 540C543.

[14] Yi Wang, Jian Yang, Weiliang Zhao, *Change impact analysis for service based business processes*, IBM Systems Journal, 2005, 44(4): 653-668.

[15] Kama, N., Azli, F, *A change impact analysis approach for the software development phase*, in Software Engineering Conference (APSEC), 2012 19th Asia-Pacific (Vol. 1, pp. 583-592). IEEE.

[16] Maskeri, Girish, Santonu Sarkar, Kenneth Heafield. *Mining business topics in source code using latent dirichlet allocation*, in Proceedings of the 1st India software engineering conference. ACM, 2008.

[17] Biggers L R, Bocovich C, Capshaw R, et al., *Configuring latent Dirichlet allocation based feature location*, Empirical Software Engineering, 2012.

[18] Kaiyuan Li, Zhensheng Xu, Xiangping Chen, *A platform for searching UI component of android application*, in ICDH 2014, Nov. 28-30, 2014, Guangzhou, P. R. China.

Learning Folksonomies for Trend Detection in Task-Oriented Dialogues

Gregory Moro Puppi Wanderley

Postgraduate Program in Informatics - PPGIA
Pontificia Universidade Católica do Paraná - PUCPR
Curitiba, Brazil
gregory@ppgia.pucpr.br

Emerson Cabrera Paraiso

Postgraduate Program in Informatics - PPGIA
Pontificia Universidade Católica do Paraná - PUCPR
Curitiba, Brazil
paraiso@ppgia.pucpr.br

Abstract— Dialogues are created by the interaction between people, who speak different kinds of topics using natural language. Task-oriented dialogue aims the solution of a given task in a given domain. Folksonomies are knowledge structures composed of users, tags and resources. Folksonomies emerge from the tagging process in collaborative tagging systems. Dialogues and folksonomies have in common their social dimension. One of the main characteristics of the folksonomies is its social dimension (users), which is also presented in dialogues, through the interaction between human beings. In this research, we describe a method that performs the learning of folksonomies, represented by a quadripartite model, from task-oriented dialogues. Using the learned folksonomies, we propose an approach for trend detection (those topics being discussed more than others). The main difference from others approaches is that we use the content of each resource in this process. This can be useful for instance, to retrieve the topics addressed by the interlocutors of the dialogues, in different time intervals. Experiments with a real-world task-oriented dialogue corpus were done.

Keywords - Folksonomies, Dialogue, Trend Detection.

I. INTRODUCTION

Dialogue is essentially the interaction between speakers and listeners, called interlocutors, composed of utterances. Among the types of dialogues that exist, task-oriented dialogue aims the solution of a given task in a given domain. Such dialogue brings the concise sequence of the solution of a task, based on the request of someone in order to accomplish something, until the solution given by another interlocutor, which may be used to determine the solution path of that task. Task-oriented dialogues have two kinds of interlocutors (see Table I for an example), one asking for help (named *user* in this research) and another with the knowledge of the domain (the *attendant*), aiming to support the former in solving the task. For Traum and Hinkelman [1], one of the main characteristics of task-oriented dialogues is the dissemination of knowledge, i.e., the interlocutor with more knowledge transfers it to the one asking for help [2].

Folksonomies are structures of knowledge representation that emerge from the tagging process in collaborative tagging systems [3]. The tagging process corresponds to the assignment of tags to resources by users. Thus, folksonomies are composed by users, tags and resources. Resources can be any object that users are interested in tag, such as photos and videos. One of the main characteristics of the folksonomies is its social dimension (users), which is also presented in dialogues, through the interaction between human beings.

TABLE I. EXCERPT OF A TASK-ORIENTED DIALOGUE.

<i>Interlocutor</i>	<i>Utterance</i>
u_1	Hello, I'd like to ask a question.
a_1	Of course, go ahead!
u_1	How many years of work I need to have in order to ask for retirement?
a_1	According to the constitution, 35 years for a men or 30 years for women.

This research introduces a method to perform the learning of folksonomies from dialogues. We intent to retrieve information from the dialogues, for instance, the topics addressed by people in different time intervals. Trending topics are those topics being discussed more than others.

This article is organized as follows: Section 2 presents the concept of folksonomy. Section 3 describes the method to obtain a folksonomy from dialogues. Section 4 presents our proposed approach. Experiments and results obtained are shown in Section 5. Finally, conclusions and future work are presented in Section 6.

II. FOLKSONOMIES

Collaborative tagging systems are characterized by the idea of tagging resources or objects through terms or keywords (tags). Such terms are freely created by different users in their own words and serve as reference for a particular resource or object of their interest. Examples of tagging systems and their resources include Delicious (URLs), Flickr (pictures), and last.fm (music). In such systems, users tag resources (URLs, pictures, or music) in order to describe or categorize them [4]. According to [5], tagging systems offer benefits including

¹ (DOI reference number: 10.18293/SEKE2015-098)

future information retrieval, contribution and sharing, task organization, expression of opinion, among others.

The structure of knowledge representation that emerges from the tagging process is called a folksonomy [3]. According to Thomas Van der Wal [6], who coined the term “folksonomy,” the word is a portmanteau of “folk” and “taxonomy,” i.e., taxonomy created by the people.

Folksonomies can be defined using a formal and well-accepted model called the “tripartite model” [7], compounded by three entities – users, tags, and resources – beyond a relationship that connects them. Based on the approach suggested by Schmitz and his colleagues [8], a folksonomy can be defined as a tuple $\mathbb{F} := (U, T, R, Y)$, where: U, T, R are the finite sets of users, tags and resources, respectively, and Y is the ternary relation between them, i.e., $Y \subseteq U \times T \times R$. This relation is also called “Tag Assignment.”

The “personomy” P_u of some user $u \in U$ is the restriction in \mathbb{F} for u , i.e., $P_u := (T_u, R_u, I_u)$ with $I_u := \{(t, r) \subseteq T \times R : (u, t, r) \in Y\}$. The personomy of a user corresponds to the set of all tag assignments that he/she has generated while tagging a given domain. Based on this, we can infer that a folksonomy is the union of all personomies of all users who participated in tagging the domain in question.

Computationally, folksonomies can be represented by a tripartite graph $G := (V, E)$ composed of users, tags and resources [9]. This graph has the following characteristics:

- The set V of vertices is formed by the three entities users, tags and resources, that is, $V := U \cup T \cup R$;
- An edge $e \in E$ (set of edges) connects two nodes, only if exists a Tag Assignment (a user has assigned a tag to a resource) that correlates them:
 - $\forall u, r \in E_T (u, r) \rightarrow \exists t \in Y (u, t, r)$ (a tag linking a user to a resource)
 - $\forall u, t \in E_R (u, t) \rightarrow \exists r \in Y (u, t, r)$ (a resource linking a user to a tag)
 - $\forall t, r \in E_U (t, r) \rightarrow \exists u \in Y (u, t, r)$ (a user linking a tag to a resource)

So, $E := E_T \cup E_R \cup E_U$.

Figure 1 shows an example of folksonomy. The ternary relationship Y between the entities is represented by the lines connecting them.

The fact that two any tags often appear together tagging the same resources is a sign of the existence of a relationship between them. Thus, in a folksonomy it is possible to associate its tags, such as using the number of resources they have tagged together [10]. In this case, two any tags t_A and t_B have a relationship b between them if and only if they have appeared together (tagging the same resources) at least x times. Moreover, x can be considered to be the weight w in this relationship. Formally, the sentence that defines the existence of the co-occurrence relationship between two tags is given by: $\forall u, r, t_A, t_B | (u, t_A, r) \in Y \wedge (u, t_B, r) \in Y \rightarrow b(t_A, t_B) \wedge t_A \neq t_B$.

III. THE LEARNING METHOD

In this section we present a method for learning folksonomies from task-oriented dialogues. In order to explain better our approach, firstly we present an extension of the formal definition of the tripartite model of Folksonomies.

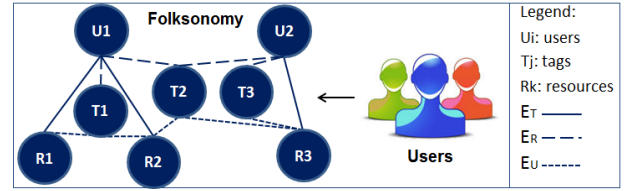


Figure 1. An example of folksonomy.

A. Formal Definition of Folksonomy Obtained from Task-oriented Dialogues

We represent users, tags, and resources of folksonomies as follow: users are “attendants” of task-oriented dialogues, resources are the utterances of attendants, and tags are the nouns of these utterances. Tagging is implicitly carried out according to our conception, i.e., tags assigned to resources are obtained from utterances generated in dialogues. Nevertheless, these utterances (resources) and, consequently, tags (nouns) are created, in this case, by the interlocutors of dialogues. We have chosen to use attendants as the users of folksonomies and their utterances as the resources because we assume that attendants have complete knowledge of a given domain. By contrast, interlocutors of type “user” need help to solve a problem or carry out a task.

According to [11], in order to refer and distinguish between objects, humans use “nouns.” This is one reason for using only nouns (instead of verbs, etc.) of the attendants’ utterances as tags of the folksonomies. Furthermore, in collaborative tagging systems, users typically use nouns to represent objects, such as “house,” “airplane,” and “violin.” According to [12], in the Delicious system, objects represent the vast majority of tag assignments performed by users and account for 76% of all tags. In terms of nouns as the grammatical class of tags used, this percentage is still higher at 88%.

Now, we present the necessary definitions related to a folksonomy learned from task-oriented dialogues:

Definition 1. A subset of users l belongs to a given attendant a and is composed of all users with whom he/she has dialogued in a given domain. Each attendant has one, and only one, subset of users. Formally, let

- A be the finite set of attendants (a be an attendant belonging to A);
- U be the finite set of users (u be a user belonging to U);
- D be a dialogue corpus (d be a dialogue belonging to D);
- Du be a function $Du: A \times D \rightarrow U$ that returns the user attended by an attendant in some dialogue;
- Ut be the set of utterances of all dialogues.

The subset of users for the attendant a (a is a constant) can then be defined by the predicate $l: \forall d ((a, d) \in A \times D) \rightarrow l(Du(a, d))$.

Definition 2. Formally, a folksonomy obtained from task-oriented dialogues can be defined as a tuple $\mathbb{F} := (A, T, R, U, Y')$, where

- A is the finite set of the users of the folksonomy. That is, the attendants of the task-oriented dialogues (who have full knowledge of a given domain);
- T is the finite set of tags, which are the nouns of the utterances that attendants have generated in the dialogues;
- R is the finite set of resources of the folksonomy, and consists of the attendants' utterances;
- U is the finite set of users;
- Y' is the quaternary relation among the above, i.e., $Y' \subseteq A \times T \times R \times U$. This relation is also called "tag assignment."

Thus, a folksonomy obtained from task-oriented dialogues is represented by a "quadrupartite model" in that it has four dimensions – attendants, tags, resources, and subsets of users – in contrast to the three dimensions of the tripartite model (users, tags, and resources).

The personomy P_a of a given attendant $a \in A$ is the restriction in \mathbb{F} on a , i.e., $P_a := (T_a, R_a, I_a, I_a)$ with $I_a := \forall t, r, u I_a(t, r, u) \rightarrow (a, t, r, u) \in Y'$. Intuitively, the personomy of a given attendant corresponds to the set of all tag assignments obtained from utterances produced by the attendant. Based on this, we can infer that a folksonomy is the union of all personomies of all attendants who have participated in the dialogues of a given domain.

We also adopted the notion of "relationship between its tags" as described in Section 2. Any two tags t_A, t_B of a folksonomy will have a relationship $b \in B$ (set of relationships between tags) between them if and only if such tags appear together (tagging the same resources) at least x times. Formally, this is given by the sentence: $\forall a, u, r, t_A, t_B b(t_A, t_B) \rightarrow ((a, t_A, r, u) \in Y' \wedge (a, t_B, r, u) \in Y' \wedge t_A \neq t_B \wedge w(t_A, t_B) \geq x)$.

The weight w adopted in this research for the relationship between two tags is the number of dialogues in which the relevant tags have appeared together. It is important to note that for two tags to be considered as appearing together does not require that they be in the same utterance in a given dialogue. These tags may belong to different utterances, but must belong to the same dialogue. The weight w of the relationship between two tags t_A and t_B belonging to T can be defined as a function $w: T \times T \rightarrow \mathbb{N}$, where \mathbb{N} is the set of natural numbers.

B. Learning Folksonomies

The method of learning consists of two steps: preprocessing and learning. It is important to note that this method is based on the principle that utterances in dialogues are identified in the dialogue corpus according to type of the interlocutor (attendants or users) that have generated them. It does not require or use information regarding people (attendants/users) that have generated the dialogues of the corpus.

Firstly, the preprocessing activity receives the dialogue corpus as input and makes it fit for use in the remainder of the process. As shown in Figure 2, the steps that compose preprocessing are "Extract Attendants' Utterances," "Extract Nouns," and "Remove Duplicate Nouns."

"Extract Attendants' Utterances" receives the dialogue corpus and extracts only the attendants' utterances from it. The main purpose of this "filtering" is to forward to the subsequent

steps of the method only utterances that represent the relevant domain.

Formally, we can represent obtaining the set of the attendants' utterances as an unary predicate $Ut = \forall a, d, ut Ut(ut) \rightarrow (a, d, ut) \in A \times D \times Ut$, where ut is an utterance of the attendant a in a dialogue d belonging to the dialogue corpus D .

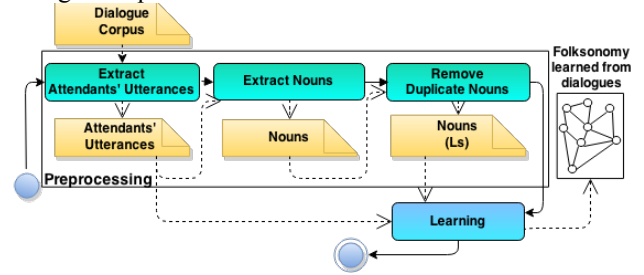


Figure 2. The Preprocessing step.

The next step is "Extract Nouns," which extracts the nouns from the attendants' utterances obtained in the previous step. The purpose of this extraction is to initiate the process of obtaining the nouns that are later converted into tags of the learned folksonomy. The nouns in the attendants' utterances of Ut are identified by a morphological analysis using a parser. Formally, the nouns extracted from Enu can be represented by a multiset (which admits repetitions in its elements) $S := \{\infty, sub : (sub \in ut) \wedge (ut \in Ut)\}$, where sub represents the nouns in the utterances of the attendants.

"Remove Duplicate Nouns" eliminates repetitive nouns from S . The final output of this step, and of the preprocessing stage, is a set Ls of unique nouns. Formally, Ls can be represented by the set $Ls := \{sub : (sub \in S)\}$, where sub is a noun of the multiset S .

The "Learning" activity builds a folksonomy automatically from the dialogue corpus. It consists of the following steps: "Obtain Folksonomy Tags," "Obtain Folksonomy Resources," "Obtain Relationships between Tags," "Obtain Attendants of the Folksonomy," "Obtain Users of the Folksonomy," and "Generate Folksonomy," as shown in Figure 3.

"Obtain Folksonomy Tags" selects nouns from Ls as candidates for tags of the folksonomy. For this, the method performs a "ranking of nouns." The aim of this ranking is to obtain the inverse document frequency (IDF) [13] of each noun of Ls in the dialogues of the dialogue corpus. The nouns ("sub") with IDF values (called "IDFsub") below a threshold frequency fc_l (see (1)) are discarded. In the context of this research, the IDF represents the importance of each noun of Ls in the dialogue corpus. Moreover, we assume that nouns that have a lower value (are less important) than the threshold represented by fc_l should not be part of the given domain. Thus, if those nouns are incorporated into the folksonomy as tags, the representation of the domain will be divergent. The nouns that are retained after applying fc_l are considered part of the domain and are tags of the set T of tags of the folksonomy. Formally the set T can be represented as: $T = \{sub : (sub \in Ls) \wedge (IDFsub \geq fc_l)\}$.

$$fc_l = \frac{\sum_{i=1}^{|Ls|} IDFsub_i}{|Ls|} \quad (1)$$

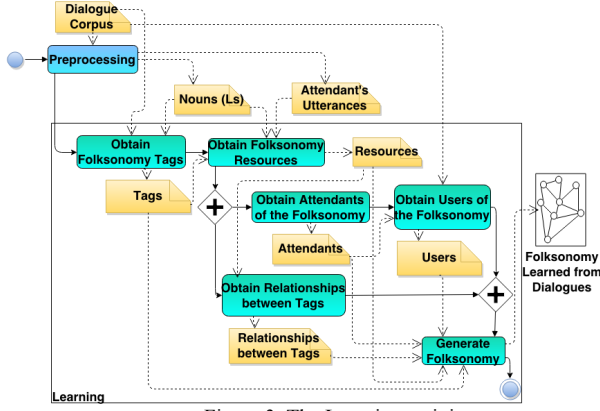


Figure 3. The Learning activity.

“Obtain Folksonomy Resources,” obtains the resources (attendants’ utterances) of the folksonomy that is being learned. To select the attendants’ utterances “ ut ”, which will subsequently become resources, we use tags of the set T (output of the previous step). Given that these tags are nouns belonging to the given domain, we count the number of nouns (“sub”) of a given utterance that belong to T , i.e., are tags. The purpose of this is to verify the attendants’ utterances that belong to the domain so that these can be adopted as resources. For each utterance, it calculates a Ratio of Inclusion p_{ut} (see (2)). This ratio measures the percentage of nouns of an utterance that are tags of the folksonomy, i.e., those belonging to the domain. Utterances with a Ratio of Inclusion p_{ut} , greater than or equal to p_1 (see (3)) are adopted as folksonomy resources. It is important to note that, we calculate the Ratio of Inclusion only for utterances containing more than one noun. This is because utterances containing only a noun may be generic and therefore may not add any knowledge to the given domain.

$$p_{ut} = \left(\frac{|\{sub : (sub \in ut) \wedge (sub \in T) \wedge (ut \in Ut)\}|}{|\{sub : (sub \in ut) \wedge (ut \in Ut)\}|} \right) \times 100 \quad (2)$$

$$p_1 = \frac{\sum_{i=1}^{|Ut|} p_{ut_i}}{|Ut|} \quad (3)$$

The step “Obtain Relationships between Tags” locates possible relationships between tags that comprise the folksonomy being learned. For this, all possible pairs of tags from the set T are first generated, i.e., a combination C_2^k , where k is the number of tags of T . For each generated pair of tags, we calculate the frequency ($fpar$), which indicates that the two relevant tags appear to tag the same resources. That is, the pairs of tags with frequencies ($fpar$) lower than $f\hat{c}_2$ (see (4)), are discarded. By contrast, pairs of tags with frequencies greater than or equal to $f\hat{c}_2$ represent a relationship between members of the pair, and thus will be part of the set B of relationships between tags. Formally, $B := \{b\}$, where b is a relationship between two given tags, and b can be given by: $\forall a, u, r, t_A, t_B, b(t_A, t_B) \rightarrow ((a, t_A, r, u) \in Y' \wedge (a, t_B, r, u) \in Y' \wedge t_A \neq t_B \wedge fpar \geq f\hat{c}_2)$.

$$f\hat{c}_2 = \frac{\sum_{i=1}^{|C_2^k|} fpar_i}{|C_2^k|} \quad (4)$$

The “Obtain Attendants of the folksonomy” step obtains the set A of attendants of the folksonomy. For each resource of the set R , the method extracts all attendants a , and this forms the set A of the folksonomy. Formally, $A := \{a : (a \in r) \wedge (r \in R)\}$, where $r \in R$.

The step “Obtain Users of the Folksonomy” acquires the set U of users. The set U of interlocutors of type “user” (asking for the assistance of the attendants) is obtained by extracting all interlocutors of type u from the dialogue corpus used as input in the method. Formally, U is obtained as follows: $U := \{u : (u \in d) \wedge (d \in D)\}$, where d is a dialogue of the dialogue corpus D .

The last step is “Generate folksonomy,” which generates the final structure of the folksonomy. Given sets A , T , R , U , and B obtained in the preceding steps of the learning activity, the method connects the elements of these sets through the quaternary relation Y' (from Definition 2 in this section). For each element from the set A of attendants, who are the “users” of the proposed folksonomy, we extract their personomies P_a based on Y' . The set of personomies of all attendants represents the folksonomy \mathbb{F} , i.e., $\mathbb{F} := (\forall a \in A) \cup \{P_a\}$, where $P_a := (T_a, R_a, l_a, I_a)$ with $I_a := \forall t, r, u, I_a(t, r, u) \rightarrow (a, t, r, u) \in Y'$. The tags of the personomies are then connected through relationships in set B (relationships between tags).

IV. TREND DETECTION THROUGH FOLKSONOMIES

In our context, trend detection refers to retrieving topics addressed at different time intervals by interlocutors in a dialogue. Trending topics are issues that are being discussed more often than others. The topics detected in a given time interval are retrieved from a folksonomy learned from dialogues in the dialogue corpus belonging only to that particular time interval. Thus, for a sufficiently long period of time, we might have several folksonomies (each learned from dialogues within a given time interval).

Once found, each topic may be compared with topics from other learned folksonomies in order to find common elements. If a given topic appears at different time intervals, i.e., in distinct folksonomies, it can be considered a trend. In other words, this means that interlocutors of the type “user” have been addressing some topic at different time intervals. Furthermore, by ranking each retrieved topic according to the number of dialogues in which it has appeared within a particular time interval, one may retrieve the most discussed topics in a given period of time. It is also possible to verify whether a given topic has gained or lost popularity in different time intervals. This can be accomplished by checking to see whether a given topic has appeared in different folksonomies, and whether it has changed its position in the rankings of those folksonomies.

The main difference from others approaches is that we use the content of each resource in the process. The first step is to divide the dialogue corpus according to time intervals. For this partitioning, the dialogues must either contain information that identifies the period in which they were produced, or they should only be organized in chronological order inside the

corpus. The number of partitions can vary, and depends on the time period from which topics are retrieved.

Having partitioned the dialogue corpus, we use each partition as an input to build a distinct folksonomy. We then retrieve from each learned folksonomy topics that were addressed in the dialogues used to learn them. This step is shown in detail in Figure 4, which also shows the content generated by artifacts. The “Retrieve the Topics Addressed” is done using the sets T , U , and R , of tags, users, and resources, respectively. It is important to note that the set U is the result of using a characteristic of task-oriented dialogues, i.e., the interlocutor of type “user,” who looks for help to solve a given task (Section 1). For the “Retrieve the Topics Addressed,” we need three more definitions:

Definition 3. A “Tag in Focus” is a tag t of folksonomy \mathbb{F}_d , which has a number of users ($u \in U$) connected to it.

Definition 4. A “Tag of Context” is a tag t of folksonomy \mathbb{F}_d , which is connected to a given Tag in Focus through a relationship $b \in B$.

Definition 5. A “Topic Addressed” is composed of a Tag in Focus, a Tag of Context, and resources ($r \in R$), with the following nomenclature: Tag in Focus + Tag in Context + Resource(s). These resources (i.e., utterances) are resources that both the Tag in Focus and the Tag of Context have marked together. The primary goal of the Tag of Context and the resource(s) is to contextualize the Tag in Focus, thus forming a Topic Addressed. For example, in the airline domain, suppose that the Tag in Focus is “seat,” its Tag of Context is “reservation,” and resources are available to help contextualize them. According to the definitions, the Topic Addressed would be “seat + reservation + resources.” This indicates that users have addressed “seat reservation” in the relevant dialogues.

The Topics Addressed are extracted from a given learned folksonomy in a list h_1 with all the tags ($t \in T$) that it contains. The tags of h_1 are in descending order according to the number of users ($u \in U$) connected to each. In this study, we define interlocutors of type “user” as unique, i.e., each dialogue features a distinct user. It is possible to infer that the number of users connected to some tag is the number of dialogues in which the tag has appeared, with the tag at the top of h_1 being the most used in distinct dialogues. This is to prepare the Topics Addressed for ranking, so that the topics at the top are the most addressed. These tags are named Tags in Focus (Definition 3).

The next step in extracting a Topic Addressed is to obtain the Tags of Context of tags in h_1 . This is because if the topics were formed only by the Tags in Focus, they would not accurately describe the Topics Addressed. For example, in the context of human resources, a subject formed only by the Tag in Focus “month” may be related to various topics, such as month of vacation, month of retirement, etc. However, it would not be possible to know which of these topics it would be referring to. Thus, given a Tag in Focus of h_1 , the learned folksonomy can help verify the tags ($t \in T$) that have a relationship ($b \in B$) with this Tag in Focus. The tag that has the relationship with the highest weight with this Tag in Focus will be its Tag of Context. The Tags in Focus and their Tags of Context are stored in list h_2 .

Following this, for each element in h_2 , we retrieve all resources that a given Tag in Focus and its Tag of Context have tagged together. The output is a temporary list of Topics Addressed. The last step in obtaining a Topic Addressed is to remove its duplicates.

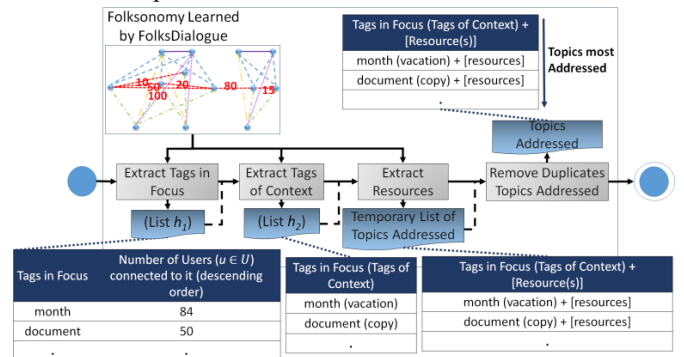


Figure 4. Retrieving Topics Addressed in the trend detection approach.

Once we have obtained the Topics Addressed from each learned folksonomy, the last part of the trend detection method involves verifying the Topics Addressed that have trended over a given time period. Each of the Topics Addressed of a given folksonomy is compared with the Topics Addressed of other folksonomies to see if it appears in them. If a Topic Addressed appears more than once over the given time, it is considered a trend.

V. RESULTS

In order to test our approach we used a dialogue corpus obtained from a City Hall in Paraná, Brazil. It is composed by 901 real task-oriented dialogues written in Brazilian Portuguese from 2006 to 2009. The 901 dialogues consisted of 7,064 utterances involving five interlocutors of type “attendant” and 901 interlocutors of type “user.” Since the users are not identified, we suppose that each dialogue involves a different user. The domain is related to human resources. The interlocutors dialogued on issues including retirement, rights of general order, probation, and vacations. Trend Detection Experiment

In order to test the trend detection method, the corpus was first split into “time intervals.” The corpus was arbitrarily chosen to be divided into eight equal parts or eight “time intervals,” each representing six months of the corpus. Each of the eight parts was used as an input to the method and generated a distinct folksonomy. The Topics Addressed in each folksonomy were then retrieved, as shown in Table II. A domain expert validated all the Topics Addressed by analyzing whether they were actual topics from the dialogues. For each Topic Addressed the domain expert validated if its resources are related to its Tag in Focus and Tag of Context.

The number of Topics Addressed for each folksonomy varied because each folksonomy was learned from different dialogues taking place at different time intervals. In each period, the attendants that produced the dialogues were different and, consequently, their manner of uttering sentences was distinct.

“Folksonomy II” had only one Topic Addressed, likely because of the manner in which attendants uttered their

sentences in that particular time interval. For instance, comparing “Folksonomy I” with “Folksonomy II,” the former is composed of 192 tags and 106 resources and the latter of 168 tags and a mere 64 resources. This is because the terms used by the attendants in the utterances used for the learning of “Folksonomy II” were not considered important by the IDF (by the Obtain folksonomies Tags step). The likelihood of some utterance becoming a resource in a folksonomy is small when it has few tags and, consequently, the probability of a relationship ($b \in B$) between two tags is small as well.

Following the retrieval of the Topics Addressed for all time intervals, we looked for possible trends in these intervals, i.e., whether a Topic Addressed appeared in different time intervals.

TABLE II. TOPICS ADDRESSED RETRIEVED FROM FOLKSONOMIES.

<i>Folksonomy</i>	<i># of Topics Addressed</i>	<i>Folksonomy</i>	<i># of Topics Addressed</i>
<i>I</i>	11	<i>V</i>	39
<i>II</i>	1	<i>VI</i>	67
<i>III</i>	14	<i>VII</i>	49
<i>IV</i>	34	<i>VIII</i>	63

We found that 39 Topics Addressed were repeated over time. Table III shows a few of the Topics Addressed that became trends. The Topic Addressed “Registration” + “Problem” + “resource(s)” appeared in three time intervals (represented by Folksonomies I, III, and VII). This means that in dialogues, users reported registration problems in the first six months of 2006 (dialogues of Folksonomy I), 2007 (Folksonomy III), and 2009 (Folksonomy VII). This Topic Addressed could be useful to advise someone of the recurring problem. Another example is the Topic Addressed “Classification” + “Career” + “resource(s)”, which is about classification in the process of admission in the enterprise of the given human resource. This topic appears in the first semesters of 2007 and 2009, which are probably the periods of the selection for new employees.

TABLE III. AN EXCERPT OF TOPICS ADDRESSED RETRIEVED FROM FOLKSONOMIES.

<i>Trend (Topic Addressed)</i>	<i>Folksonomies Containing Trends</i>
“Registration” + “Problem” + “resource(s)”	I, III, VII
“Son” + “birth” + “resource(s)”	IV, VIII
“Classification” + “Career” + “resource(s)”	III, VII
“Test” + “Application” + “resource(s)”	VI, VII

VI. CONCLUSION AND FUTURE WORK

In this research we propose a method to perform the learning of folksonomies, from task-oriented dialogues, represented by a quadripartite model. Computationally, the folksonomies generated by the proposed method are represented by graphs. We also proposed an approach for

trend detection, which can be useful, for instance, to retrieve the topics addressed by the interlocutors of the dialogues, in different time intervals.

Through an experiment with a real-world task-oriented dialogue corpus, we could see that it is possible to retrieve information and detect trends over time in a dialogue corpus.

In the near future, we intend to deal with some natural language enhancement, such as abbreviations and correcting spelling errors. Even if we do not found a different dialogue corpus to test our approach, we intend to do so. Moreover, in relation to the trend detection approach, a future work that can be done is a *concept drift* [14] study. Given the fact that there is no guarantee about the behavior of users in the dialogues and consequently stability in the extracted trends (as they can change at any moment of time), this may result in inconsistencies in folksonomies learned with data from different periods of time. Thus, it may be important to study the problem and techniques to identify concept drift in order to avoid such inconsistencies.

REFERENCES

- [1] D. R. Traum, E. Hinkelman, “Conversation acts in task-oriented spoken dialogue.” *Computational Intelligence. Special Issue on Non-literal Language*, 8(3), 1992.
- [2] J. Carletta, A. Isard, S. Isard, J. C. Kowtko, G. Doherty-Sneddon, and A. H. Anderson, “The reliability of a dialogue structure coding scheme.” *Computational Linguistics*, 23(1), 1997, pp. 13–31.
- [3] I. Peters, “Folksonomies: Indexing and retrieval in Web 2.0,” De Gruyter Saur, ISBN-10: 3598251793, 2009.
- [4] C. Körner, D. Benz, A. Hotho, M. Strohmaier, and G. Stumme, “Stop Thinking, Start Tagging: Tag Semantics Emerge from Collaborative Verbosity.” In *Proc. of the 19th International Conference on World Wide Web*, 2010, pp. 521–530.
- [5] M. Gupta, R. Li, Z. Yin, and J. Han. “Survey on social tagging techniques.” *SIGKDD Explor.*, vol.12, 2010, pp. 58-72.
- [6] T. Van der Wal, “Folksonomy Coinage and Definition”. *Website*: <<http://vanderwal.net/folksonomy.html>> Accessed: 10 mar. 2015.
- [7] P. Mika, “Ontologies are us: A unified model of social networks and semantics.” *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(1), 2007, pp. 5–15.
- [8] C. Schmitz, A. Hotho, R. Jäschke, and G. Stumme, “Mining Association Rules in Folksonomies.” In *Lecture Notes in Computer Science - The Semantic Web: Research and Applications*.Vol. 4011, 2006, pp.411–426.
- [9] S. Chojnacki, and M. Klopotek, “Random graph generative model for folksonomy network structure approximation”. *Procedia Computer Science*, 1(1), 2010, pp. 1683-1688.
- [10] G. Belgeman, P. Keller, and F. Smadja, “Automated Tag Clustering: Improving search and exploration in the tag space.” In *Collaborative Web Tagging Workshop at WWW’06*, 2006, pp. 15-33.
- [11] D. W. Embley and B. Thalheim, “Handbook of concept modeling: Theory, practice, and research challenges.” *Springer*, ISBN: 978-3-642-15865-0, 2011, p. 226.
- [12] L. Spiteri, “The Structure and form of folksonomy tags: The road to the public library catalogue.” *Information Technology and Libraries*, 27(3), 2007, pp. 13-25.
- [13] G. Salton, and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management* 24, 5, 1988, pp. 513–523.
- [14] A. Tsymbal, “The problem of concept drift: definitions and related work.” *Technical Report TCD CS-2004-15*, Computer Science Department, Trinity College, Dublin. 2004.

Towards Automatic Requirements Elicitation from Feedback Comments: Extracting Requirements Topics Using LDA

Hitoshi Takahashi Hiroyuki Nakagawa Tatsuhiro Tsuchiya
Graduate School of Information Science and Technology
Osaka University
1-5 Yamadaoka, Suita, 565-0871 Japan
{t-hitosi, nakagawa, t-tutiya}@ist.osaka-u.ac.jp

Abstract

Feedback comments, such as mailing lists and reviews, contain beneficial suggestion for software developers. Recently, developers have received more and more feedback comments; but it is still difficult to extract beneficial comments from a large amount of e-mail message or reviews. Latent Dirichlet Allocation (LDA) is a promising way of topic modeling, which classifies documents according to implicit multiple topics. In this paper, we tried to apply a requirements elicitation based on LDA to two different sources, i.e., Apache Commons User List and App Store reviews, and discuss the feasibility of this approach. An interesting finding was that some usual stop words indicated requirements description. This suggests that these words should be removed from the stop word list before applying LDA.

1 Introduction

Feedback comments are beneficial resources for developers because these comments contain information about what they want. Recently, more and more feedback comments have been gathered due to the improvement of user's environment. As stated in [5], more than one million of reviews in Google Play are uploaded per a day.

A lot of feedback comments can be useful as references for development; however, we usually have to manually extract beneficial data such as implicit requirements from these resources. The size of the resources is often too large to deal with manually.

Automatic extraction of requirements from user's feedback comments that are described in natural languages would be desirable. Some applications of linguistic engineering technology to manage requirements in mass software development have received recent attention. Dag et al.

[7][10] introduced the Baan requirements management process, which finds the relationships between feedback comments and business requirements (objectives). Some studies addressed the extraction of requirements from feedback review comments of smartphone applications. Fu et al. [8] introduced a system that analyzes App Store reviews and identifies problems by topic modeling. The system also classifies feedback comments into individual functions. Guzman et al. [9] proposed an approach that introduces rating of each function from words and emotions associated with them. The latter two studies use Latent Dirichlet Allocation (LDA) [6] for topic modeling. These studies extract topics related to functions of the target application; however, to the best of our knowledge, there is no existing work for automatic requirements elicitation from the feedback comments.

As the first step of the automatic requirements elicitation from feedback comments, we introduce a requirements elicitation process from the feedback comments based on LDA topic modeling. We also apply this elicitation process to Apache Commons User List and App Store reviews, which are respectively of types e-mail messages and reviews, aiming to elicit topics related to requirements. The experimental results indicated that our approach still needs further improvement; however, the results also provided some finding about the requirements elicitation from the feedback comments.

This paper is organized as follows: Section 2 describes the research questions in this work. Section 3 explains our requirements elicitation process from the feedback comments. Section 4 presents the results of experimental elicitation from two different types of feedback comments. Section 5 discusses the feasibility of our approach, and Section 6 concludes the paper.

Table 1. Target resources.

Resource	E-mail	Review
Language	English	English
Main subject	technical questions, announcement	evaluation (rating), bug report
Size	relatively large	usually small
Frequency of requirements	few	many

2 Research Questions

Our goal is to establish automatic requirements elicitation from feedback comments. In particular, we address the following two research questions.

- **RQ1:** Can we classify feedback comments into those that include requirements and those that do not include?
- **RQ2:** Does the quality of extracted requirements vary depending on the types of the feedback comments?

We use LDA for the requirements elicitation from the feedback comments. LDA constructs topics from the resource documents, i.e., feedback comments in this study. These topics indicate implicit characteristics of the documents and enable to classify documents. Therefore, RQ1 corresponds to the question whether we can extract topics related to requirements description.

As for RQ2, feedback comments can be roughly classified into e-mail messages and reviews. Table 1 shows their characteristics. In this study, we try to clarify whether the performance of the LDA classification depends on the resource type, i.e., e-mail messages and reviews.

3 Elicitation Process

Figure 1 illustrates our elicitation process. This elicitation consists of four activities, *lemmatizing*, *topic modeling using LDA*, *requirements topic elicitation*, and *requirements comments extraction*. The following sections explain these activities.

3.1 Preparation: Lemmatizing

Feedback comments that we deal with in this study are written in English and contain verb words that are conjugated. For example, “wishes” is the third person form of “wish” and they have same meaning but LDA recognize them as completely different words. We lemmatize conjugated with the WordNet Lemmatizer [11][4], which can be used to remove affix from the input word.

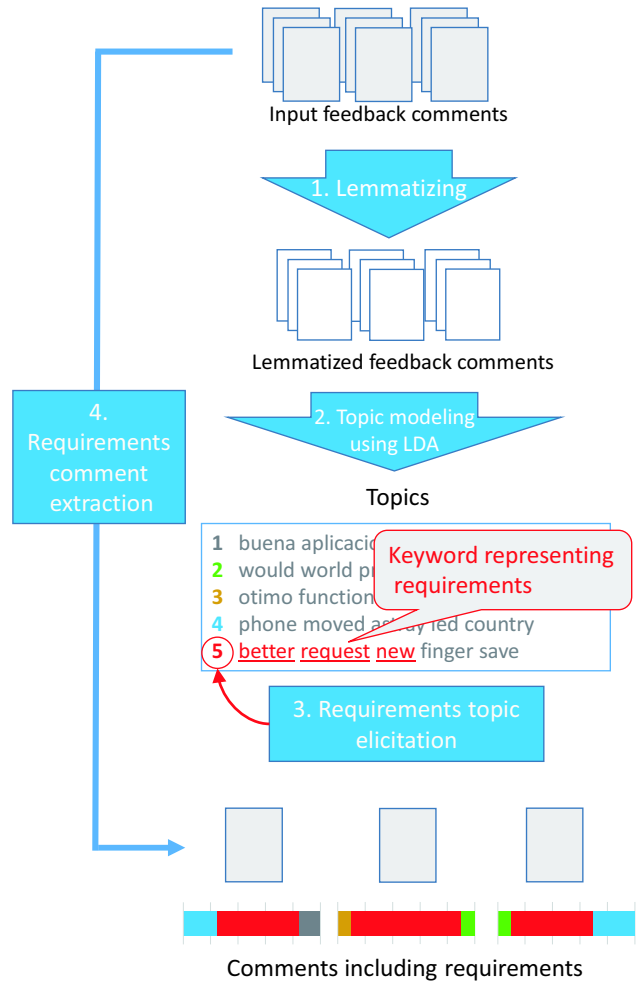


Figure 1. Elicitation process.

3.2 Topic Modeling Using LDA

We use Latent Dirichlet Allocation (LDA) [6][12] for topic classification. LDA is a topic model and widely used for unsupervised word classification. In LDA, topic distribution generates the topic for a word, and the topic for a word generates the specific word. Figure 2 presents the graphical model of LDA. The symbols in the figure correspond to the following concepts:

ϕ : word distribution for topic

θ : topic distribution for document

z : topic for word

w : word

α, β : hyper parameter. These parameters are given or usually estimated by a machine learning tool.

K : the number of topics

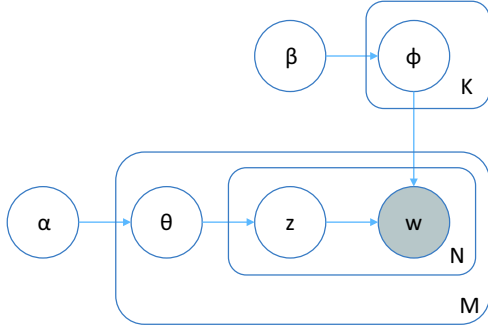


Figure 2. Graphical model representation of LDA.

M : the number of documents

N : the number of words in m th document

Hyper parameter α determines the topic distribution for document θ , and the topic z is determined according to θ . Another hyper parameter β determines the word distribution for topic ϕ , and finally the word w is determined according to z and ϕ .

In order to construct the topic model that can classify the feedback comments, we use LDA in this study according to the following steps:

Step 0. (Preparation) Give a set of documents (M and N are determined) and set the number of topics K .

Step 1. Set the default topic for each word in all comments.

Step 2. Select each word w from the comments.

Step 3. Change the topic z for the word w according to the probability P shown in Eq. (1).

Step 4. Repeat 2. and 3. until N_t^- and N_{mt}^- in Eq. (1) are converged.

Step 5. Output ϕ as the word distribution for topic and θ as the topic distribution for comment.

$$P(z = t | Z^-, W, \alpha, \beta) \propto \frac{\beta + N_{tw}^-}{\beta V + N_t^-} (\alpha_k + N_{mt}^-) \quad (1)$$

Z^- : set of topics of all words excluding the word w .

W : set of all words in all documents.

N_t^- : the number of words in all documents whose topics are t .

N_{tw}^- : the number of word w in all documents whose topic is t .

N_{mt}^- : the number of words in selected document m whose topics are t .

α_k : the k th (topic k 's) parameter α .

V : the number of words in all documents.

We use MALLET [3], a tool package for the topic classification based on LDA. This tool also has the word tokenization and unnecessary word removal functions. We input feedback comments to MALLET, and MALLET outputs ϕ , θ , and z by constructing the topic model based on LDA.

3.3 Requirements Topic Elicitation

After the topic model is constructed by MALLET, we find the topics related to the requirements by using ϕ . We identify the words likely to be contained in the requirement description and find topics that contain many words related to the requirements description as the topics related to the requirements.

3.4 Requirements comment extraction

We can extract possible comments that include requirements description based on the topics related to the requirements acquired in the previous activity. We use θ and find the comments that has a high relationship to the topic related to the requirements.

4 Experiment

4.1 Elicitation from Different Resources

We applied our elicitation process to two different sources, i.e., Apache Commons User List and App Store reviews. Apache Commons User List [1] is a mailing list for contacting users and developers, supported by Apache Software Foundation. Most of the messages describes technical questions and answers about software belonging to apache commons. Other mails are for the announcement of the latest version and questions about software functions; therefore, there are few messages related to requirements for new functions. Figure 3 illustrates an example e-mail message that contains the requirement for adding new output option to the CSVPrinter class.

App Store and Google Play are well-known review platforms. In this experiment, we use reviews in App Store [2] as a resource of reviews. The reviews are composed of five parts: *title*, *rating*, *author*, *date*, and *body text*.

The review shown in Figure 4 is a review for Google Maps. This review requests an additional function to rename designated places to Google Maps.

Subject: [CSV] Wish: format-specific date generation

Hi -

It would be useful if printing a Java Date or Calendar to a CSVFormat. EXCEL CSVPrinter would generate output that Excel recognises as a date-and-time. For example the following

```
PrintWriter outputWriter = new PrintWriter(new File-
OutputStream("output.csv"));

...

which Excel only treats as a string. (It will recognise
e.g. yyyy/mm/dd as a date but I wouldn't know where
to look for a definitive set of formats it will consume.)
Ditto probably printing Calendar.getInstance(), or the
new Java 8 LocalDate etc. classes.


One argument against though is then the library per-
haps ought to do the reverse, i.e. spot that it has been
passed a date in and construct a Date class for the value
at parse time which may be expensive and often unnec-
essary.



...


```

Figure 3. An e-mail message containing requirements.

We collected the same number of messages from Apache Commons User List and Google Maps reviews in App Store. We applied our elicitation process and extracted topics and comments related to the requirements. We, in particular, collected 300 messages and reviews from Apache Commons User List and Google Maps reviews in App Store, respectively. Among them, 17 messages contain requirements ($\#R_{max_E}$) and 47 reviews contain requirements ($\#R_{max_R}$).

We constructed topic models under the conditions that the topic size is 20, 40, and 100, respectively, where the topic size determines how many topics LDA generates. We regarded an extracted topic (T) as a topic related to requirements if it contained at least two words related to requirements. By using the extracted topics T , we extracted comments whose topic distribution for one of the extracted topics related to requirements are over 0.1 as the comments related to requirements (C).

We made an optimization that made use of the characteristics of the words related to requirements. Since we believe that the stop words of MALLET, which are the most common words and filtered out before topic model construction, include some words, e.g., *a*, *able*, *about*, *above*, and so on,

Title : Great App !

Great App ! *****
by Gold Eagle 007 - Dec 18, 2014

This app is incredible !
But it could be better
if i could rename my places

Figure 4. A review containing requirements.

able, appropriate, appreciate, asking, ask, awfully, because, better, best, cannot, can, contains, containing, contain, considering, consider, currently, could, different, enough, except, help, hopefully, if, like, need, needs, necessary, new, normally, please, shall, should, toward, towards, tries, trying, try, unfortunately, useful, want, wants, will, why, would

Figure 5. The words removed from stop word list.

related to the requirements, we excluded some words from the stop word list. Figure 5 represents the words that we excluded from the stop word list.

4.2 Experimental results

Table 2 lists the results of applying our elicitation process, where $\#Requirement\ topics$ is the number of the extracted topics related to requirements ($\#T$), $\#Extracted\ comments$ is the number of comments that are extracted by our elicitation process ($\#C$), and $\#Requirement\ comments$ is the number of the extracted comments that were indeed related to requirements among C ($\#R$).

Table 2 demonstrates that the elicitation from reviews worked more precisely than that from e-mail messages. This table also indicates that the topic size affects the precision and recall of the requirements elicitation. In this experiment, 40 topics was the most suitable size for constructing the topic model. Another finding was that the stop word list modification made improvements of precision and recall in many cases.

Figures 6 and 7 illustrate the extracted topics related to requirements in the cases of Apache Commons User List and App Store reviews with 40 topics extraction and stop word list modification, respectively. As shown in both the figures, we extracted four topics related to requirements, respectively. We regarded some words excluded from the stop word list as the words related to requirements. For example,

Table 2. Requirements elicitation results. Here, $Precision = \#R / \#C$, $Recall = \#R / \#R_{max_E}$ (for E-mail) or $Recall = \#R / \#R_{max_R}$ (for Reviews), and $F\text{-measure} = 2 \cdot Precision \cdot Recall / (Precision + Recall)$.

Resource	Topic size	Stop word	#Requirement topics (#T)	#Extracted comments (#C)	#Requirement comments (#R)	Precision	Recall	F-measure
E-mail	20	default	1	22	0	0.000	0.000	0.000
		modified	3	81	2	0.0247	0.118	0.041
	40	default	1	23	1	0.0435	0.059	0.050
		modified	4	107	7	0.0654	0.412	0.113
	100	default	2	22	0	0.000	0.000	0.000
		modified	8	96	3	0.0313	0.176	0.053
Reviews	20	default	0	0	0	0.000	0.000	0.000
		modified	5	290	41	0.141	0.976	0.247
	40	default	1	23	12	0.522	0.286	0.369
		modified	4	111	29	0.261	0.690	0.379
	100	default	0	0	0	0.000	0.000	0.000
		modified	3	30	7	0.233	0.167	0.194

Topics related to requirements

3 {evaluation integrator number problem univariateintegrator **could** fitting **need would** math **if should** case integration rule doe class gaussintegrator default }

12 {**can** scxml log **if** time scripting improvement **need** common method class **could will** default implementation simplecontext agent issue static }

22 {ascert file digester size inumdestinations node div listnodepathdata xml version rob www set **if** add **need** coefficient default true }

35 {org commons apache user mail unsubscribe help additional command wrote **can if** doe gmail **would should** issue http pm }

Figure 6. Extracted topics from Apache Commons User List (topic size = 40, using the modified stop word list).

Topics related to requirements

10 {**need why** annoying log number phone info stop ad pop business data travel broke rating chase profit stay company }

12 {ca amazing user **will** saved friendly **if** running star review le rename **could** point accurate **why** real working live }

19 {car real information live doe wonderful happy navigational available waze constantly thousand supposed cart traveler **want because** point exceptional }

33 {street view **feature** version address close found exit **able** bad **new** streetview switch number interface ruined wo half level }

Figure 7. Extracted topics from App Store reviews (topic size = 40, using the modified stop word list).

since the word “could” may express the hope, and the word “if” is used for expressing subjunctive mood, we defined both words as the words related to requirements.

5 Discussion

We will now answer to the research questions based on the experimental results, and then discuss the limitations.

As for RQ1: “Can we classify feedback comments into those that include requirements and those that do not in-

clude?”, the usefulness of the topic modeling approach depends on whether we can extract highly precise topics related to requirements. We extracted the topics that seems to be related to requirements from both e-mail messages and reviews in some cases in our experiment; however, the adequate topic extraction seemed to require certain constraints. As the findings from our experiments, the determination of topic size and the stop word list modification should be taken into account. The topic size affects the accuracy of the classification of comments. Small topic size squeezes

multiple topics into a topic; large topic size, on the other hand, constructs exceedingly decomposed topics.

We should also consider the characteristics of the words related to requirements. Words in the general topics, such as features and domains, are usually nouns and verbs. These topics are relatively easy to be extracted by using the topic modeling techniques including LDA. However, when we consider the topic related to requirements description, we have to deal with not only nouns and verbs but also auxiliary verbs, such as *can*, *will*, *could*, and *would*. Moreover, we believe that we should extract some of multiple-topic-concerned nouns and verbs, such as "need" and "like", as the words in the requirements topic. Most of these words are included in the stop word list, which is defined for elaborating topics to be extracted. Therefore, we believe that we should exclude words related to requirements from the stop word list; however, such exclusion may also inject the ambiguity of topics to be extracted.

As for RQ2: "*Does the quality of extracted requirements vary depending on the types of the feedback comments?*", as roughly summarized, the elicitation from reviews tends to score higher recall rate than the extraction from e-mail messages; however, it also tends to extract excessive quantities of comments. A possible reason is the contents size. Since review comments are generally short, the extracted topics from reviews tend to cover large amount of comments. Further precise topic classification should be required for improving the precision rate.

The experiment results demonstrate that eliciting requirements comments from e-mail messages is relatively difficult for our current process. Comparing Figures 6 and 7, the extracted topics from e-mail include words related to more decomposed features. A possible reason is the diversity of the contents. Comments in e-mail, such as Apache Commons User List in our experiment, generally contains more words than reviews, to explain more detailed situation, sometimes attaching source code, which includes the method and class name to the e-mail messages. This situation may hamper the construction of topics.

6 Conclusions

As the first step of the automatic requirements elicitation from feedback comments, we defined a requirements elicitation process from the feedback comments based on LDA topic modeling. We applied the elicitation process to two different sources, i.e., e-mail messages and reviews, and discussed the feasibility of our approach. The experimental results suggest that our current elicitation process has some limitations but also has a possibility of providing an automatic mechanism of requirements elicitation from review comments. We also found that effective requirements elicitation requires the stop word list modification. One of our

primary on-going studies is further improvement of precision and recall value of our elicitation. We plan to discuss the algorithms for the adequate topics extraction mechanism. A comment extraction mechanism from the acquired topics should be refined. We will also conduct case studies in the large amount of feedback comments to discover further findings for the elicitation. We believe that automatic processes for dealing with huge feedback such as our approach help us adapt to recent continuous software delivery.

References

- [1] Apache commons mailing lists. <http://commons.apache.org/mail-lists.html>.
- [2] App store. <https://itunes.apple.com/us/genre/ios/id36?mt=8>.
- [3] MALLET homepage. <http://mallet.cs.umass.edu/>.
- [4] WordNet A lexical database for English. <http://wordnet.princeton.edu/>.
- [5] AppTornado GmbH. Number of available android applications. <http://www.appbrain.com/stats/number-of-android-apps/>.
- [6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.
- [7] J. N. o. Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. A linguistic-engineering approach to large-scale requirements management. *IEEE Softw.*, 22(1):32–39, Jan. 2005.
- [8] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13*, pages 1276–1284, New York, NY, USA, 2013. ACM.
- [9] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 153–162, Aug 2014.
- [10] J. Natt och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. Speeding up requirements management in a product software company: linking customer wishes to product requirements through linguistic engineering. In *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pages 283–294, Sept 2004.
- [11] J. Perkins. *Python Text Processing with NLTK 2.0 Cookbook*. Packt Publishing, 2010.
- [12] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08*, pages 569–577, New York, NY, USA, 2008. ACM.
- [13] D. Ramage, D. Hall, R. Nallapati, and C. D. Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP '09*, pages 248–256. ACL, 2009.

MAX: A Method for Evaluating the Post-use User eXperience through Cards and a Board

Emanuelle Cavalcante^{1,2}, Luis Rivero¹ and Tayana Conte¹

¹USES Research Group, Instituto de Computação, Universidade Federal do Amazonas (UFAM)
Manaus, AM – Brazil

²FPF Tech, Manaus, AM - Brazil
emanuelle.cavalcante@fpf.br, {luisrivero,tayana}@icomp.ufam.edu.br

Abstract— User Experience (UX) is one of the most important attributes for the success and quality of a software product. UX explores how a person uses an application, and the emotional and behavioral consequences of such use. Although several UX evaluation methods allow gathering information on the reasons for a poor UX, some of them tend to make users feel uncomfortable, such as asking direct questions to shy users. This paper presents our proposal for evaluating UX, the Method for the Assessment of eXperience (MAX), which through cards and a board intends to motivate users to report their experience. The MAX method does not require experienced evaluators for performing the evaluation. Instead, this method is intended at software engineers willing to obtain data on UX and make users feel comfortable during the evaluation. To verify the feasibility of the MAX method from the point of view of users, we conducted a pilot study. The results showed that the MAX method has proven useful for evaluating the UX of finished or prototyped software applications. Also, we have made improvements in the method to meet users' needs when reporting their experience, and gathering data on factors affecting the UX.

Keywords: *User eXperience; Evaluation Method; Software Quality.*

I. INTRODUCTION

Usability has been considered one of the main attributes that represent quality in an instrumental, task-oriented view of interactive products [1]. However, despite the increasing attention that usability has received in the development of software applications [2], a new term, “User eXperience” (UX), has emerged as an umbrella phrase for new ways of understanding and studying the quality in use of interactive products [1].

UX is an important attribute for the success of software products, searching for new approaches for their design and evaluation, which accommodates experiential qualities of technology use rather than product qualities [3]. According to the ISO 9241 [4], user experience is defined as “*a person's perceptions and responses that result from the use or anticipated use of a product, system or service*”. Such definition is complemented by the definitions of other authors. For instance, Law et al. [5] state that UX explores how users feel about the use of a product, or in other words, the emotional and affective aspects. Thus, UX turns essential in order for a software to be accepted by its users since, besides

being usable and work correctly, the software should be emotionally appealing [6].

The increasing interest in the improvement of UX has been the motivation for the creation of new evaluation methods that allow capturing the users' emotions and the aspects that affect the reported emotions [7]. In that context, it is necessary to highlight the difference between usability evaluation methods and UX evaluation methods. Usability tests tend to focus on task performance, while UX focuses on the experience that was lived by the user and his/her emotions.

As it is subjective, UX deals with the feelings and thoughts of an individual regarding the use of a software, product or service. According to Law et al. [5], usability measures such as execution time or number of errors are not enough to measure the user experience of a product. Also, it is not always easy for a user to realize what (s)he is feeling or even express his/her user experience. Therefore, it is necessary to employ new approaches that stimulate and guide the user during the report of his/her experience. The report of the use by the user allows the evaluator to gain insights on the experienced problems and also understand positive aspects of the product use. Furthermore, it is possible to gather data regarding if the product was accepted by the user or not.

Vermeeren et al. [7] state that there is a high number of UX evaluation methods being employed by both the industry and the academy. Nonetheless, according to Miles et al. [8], some methods can cause discomfort to the users who participate from the evaluation since it forces them to report their experience. Users can feel forced to participate in the evaluation sessions and this can cause bias in the report of their experience [6]. Consequently, it is necessary to develop practical methods that are easy to use and do not feel like a chore, so the users can be comfortable when reporting their experience [7]. Also, these methods must be easy to use from the point of view of software engineers, so practitioners from the software industry are able to apply them and gather data to improve the quality of software applications under development [7].

In order to propose a UX evaluation method that is easy to use and motivates users to report their experience, we have proposed the Method for the Assessment of eXperience (MAX). MAX introduces a set of cards and a board to guide the user through the UX evaluation process. That way, the

evaluator can gather information on the user's emotions, how easy and useful it was to use the system, and his/her intention to use the system again if given the chance. The MAX method can be applied at any stage of the software development process, after the use of mockups, prototypes, or the final versions of interactive systems.

This paper presents the initial version of the MAX method, and its initial evaluation from the point of view of users through a pilot study. We evaluated MAX through questionnaires in order to gather information regarding if it was easy to apply by the users. Based on the results from the evaluation, we have made improvements in MAX, generating a second version. As a result of this application of the MAX method, we were able to evaluate its feasibility for the evaluation of UX.

This paper is organized as follows. Section 2 presents the background and related work of this research, where we provide UX definitions and a brief description of some of the proposed UX evaluation methods. Next, Section 3 describes the proposed MAX method, while Section 4 describes the execution and results from the pilot study and the initial improvements over MAX. Finally, Section 5 presents our conclusions and future work regarding this research.

II. BACKGROUND AND RELATED WORK

User eXperience involves all aspects of the user interaction and aims at guaranteeing that software systems become satisfying, interesting, useful, motivating, beautiful and adequate [9]. According to Roto [10], such experience includes the emotions, preferences, physical and psychological reactions of the user that can occur: (a) before usage, or in other words, the expectations of the user regarding the software product; (b) during usage, which is the momentary experience of the user; and (c) after usage (post-use), when it is possible to verify if the users' expectations were actually met. Furthermore, UX is a consequence of the internal state of the user (his/her expectations, needs, motivations, humor, and others), the features of the system (utility, ease of use, functionality, and others) and the context and environment in which the interaction between the user and the system occurs [11].

UX evaluation plays an important role in the development of interactive applications, since it assesses their value regarding how the users will apply, perceive, and learn the software, as well as how it will evolve and adapt to the users' changing expectations [12]. In that context, UX evaluation methods can be employed to gauge the product success in the real market and attract potential customers [13].

In order to identify which UX evaluation methods were proposed, Vermeeren et al. [7] carried out a review. Such review allowed identifying 96 UX evaluation methods that can be applied in different types of applications (desktop, Web, mobile, and others) and in different phases of the software development process (analysis, design, test, coding, and others). From the set of identified UX methods for evaluating the after usage of a system, one can name: scales, online surveys and probes. We will detail these methods and their (dis)advantages as follows.

Scales such as the Self-Assessment Manikin (SAM) [14] and the Unified Theory of Acceptance and Use of Technology (UTAUT) [15] allow evaluating the opinion of the users by measuring specific attributes related to UX. Through the SAM scale, it is possible to evaluate three dimensions: (a) Pleasure (pleasure/displeasure), (b) Dominance (in control of the situation/controlled by the situation), and (c) Arousal (calmed/excited). UTAUT, on the other hand, focuses on aspects related to technology acceptance, measuring effort, performance and facilitator conditions. Although scales demand less time to be employed and shy users may feel more comfortable when applying these methods, the collected data may not provide insights on the causes for a poor UX.

Online surveys can also be employed to gather information on UX. An example is the AttrakDiff [16], which allows several users from different locations and profiles to provide information on pragmatic and hedonic attributes of a product. The main issue with collecting data online is that the evaluator does not have control over the proper filling of the questionnaire, and cannot return to the users for further feedback.

Probes are another alternative for evaluating UX. With probes, users can be encouraged to report their experience. For instance, the Emocards [17], which are 16 drawings of faces (8 male and 8 female) illustrate different emotional responses to an evaluated product. When a user picks one (or more) of the emocards, (s)he can report his/her experience explaining the reasons that motivated him/her to choose it. However, the user may explain what (s)he thinks is important to be explained, concealing relevant information from the evaluators.

The methods cited above demand low costs in their application and are easy to use from the point of view of users or evaluators [7]. However, there are some disadvantages that can influence the results of the UX evaluation. According to Tähti and Niemelä [6], users applying methods that employ the representation of emotions may not be able to relate their emotions with the drawings for not understanding the representations. Also, Miles et al. [8] state that the majority of the UX evaluation methods might be intrusive, and that the fact of asking direct questions to the users about their emotions might make them feel uncomfortable. This can have a negative effect on the results of the UX evaluation, since users can hide vital information or express different emotions than the ones they are really feeling in order to please the evaluators. Finally, as mentioned before, some scales and forms may not provide information on the causes for choosing a specific answer, and not being able to request further information on why the user chose a specific answer may conceal relevant information for improving the UX. Considering the disadvantages of the current UX evaluation methods, it is necessary to provide new methods that meet the needs of the evaluators, make users feel comfortable, and make it easier for them to report their emotions. In order to meet these goals, we proposed MAX.

III. THE METHOD FOR ASSESSING EXPERIENCE (MAX)

The Method for the Assessment of eXperience (MAX) is a post-use method that aims at evaluating the general experience

of a user regarding an interactive application. MAX can be employed after the use of mockups, prototypes, interactive systems, or any artifact that allows user interaction.

The evaluation is performed through the use of cards and a board. The MAX cards allow evaluating the UX in terms of four categories: (a) Emotion, (b) Ease of Use, (c) Usefulness and (d) Intention to Use. These categories are similar to the evaluated aspects in technology acceptance methods, which consider usefulness, ease of use and intention [15]. As emotions are inseparable from cognition and are part of a user judgment about a system [18], we have also considered them in our method. In that context, the Emotion category focuses on the importance of the emotional aspects, since the experience considers the emotions, preferences and psychological reactions of the user [10]. To create the cards from the Emotion category, we considered the wheel of emotions by Plutchik [19]. The other three categories (Ease of Use, Usefulness and Intention to Use) were considered as they describe the necessary elements for achieving a positive UX [1]. In that context, the Ease of Use category aims at evaluating usability aspects of the application, while the Usefulness category aims at evaluating the user perception regarding how much the application contributes to the execution of his/her tasks. Finally, the Intention of Use category evaluates if the user would use or recommend the application.

Each MAX card presents an avatar to portray and express the possible reactions that a user can express regarding the evaluated system. Fig. 1 presents the avatar that we developed for one of the items of the Emotion category. We designed the avatar as a human cartoon form in order for the user to create empathy with the cards and be able to express him/herself more easily. Also, we chose to design cards that would not appear too formal, so users would not think of the evaluation as a chore, and would see it as something entertaining and comfortable to do.

The MAX deck of cards is inspired in a conventional deck of cards, where each evaluated UX category is represented by a symbol and color. Also, to allow users to express the intensity of an emotion, each card has an associated scale. These scales were developed and added to each of the items from the MAX categories in order to create the cards. Fig 1 also shows an example of the applied scales to describe the intensity of two items from the emotion category (a) Happy, positive and (b) Sad, negative. The cards that evaluate the UX in a positive way present a green scale, while the cards evaluating the UX in a negative way present red scales.

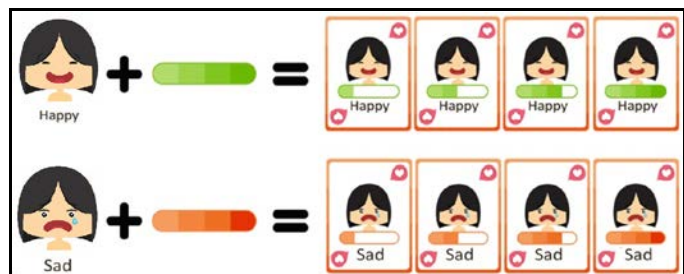


Figure 1. Avatar and intensity scales from the MAX method (v1)

Fig. 2 presents the evaluated items for each of the categories of the first version (v1) of the MAX method. At all, the MAX v1 provided 92 cards (considering all the items and their possible intensities): (a) 40 for the Emotion category, (b) 20 for the Intention to Use category, (c) 16 for the Utility category, and 16 for the Ease of Use category.

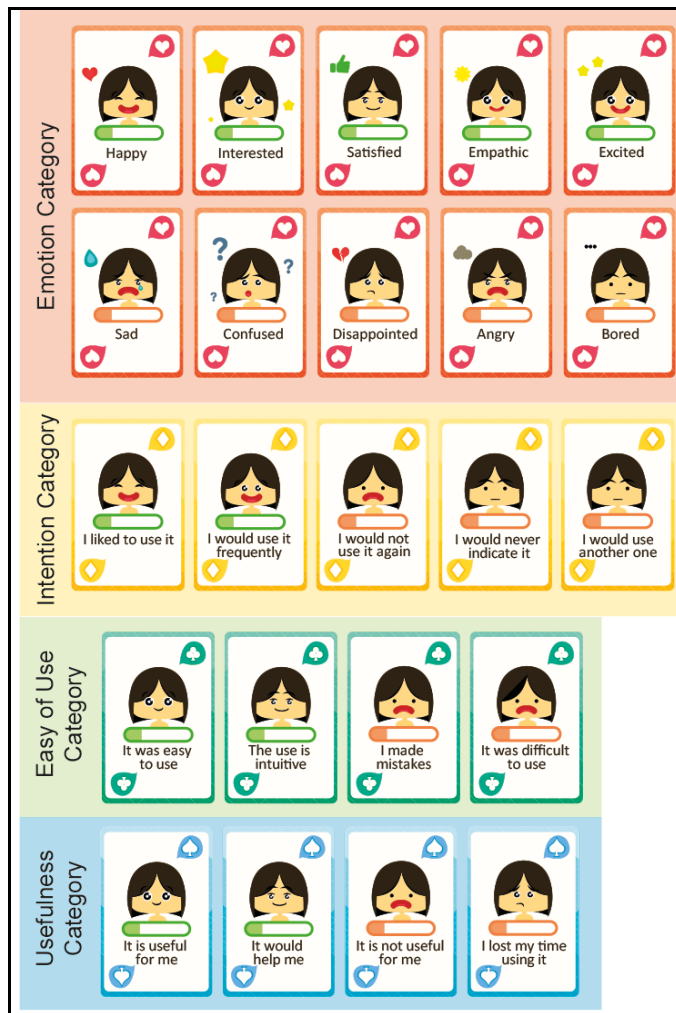


Figure 2. Evaluated items from the MAX method (v1)

In order to allow a much more dynamic application of the cards, we also developed a board. Such board shows which questions the user must answer when selecting the cards for reporting his/her experience. Initially, the board presented four questions, one for each of the evaluated categories within the MAX method: (a) What did you feel when using it? (Emotion category), (b) Was it easy to use? (Ease of Use category), (c) Do you wish to use it? (Intention category), and (d) Was it useful? (Usefulness category). Fig. 3 shows the MAX board for the first version of the MAX method.

In order to motivate users to choose at least two cards, we also added slots for each of the categories within the MAX method. These slots are spaces in which the user can place the MAX cards. We added these slots because a single card does not provide enough information regarding the user experience.

Two or more cards, on the other hand, create a sequence which makes the report clearer for the evaluator.

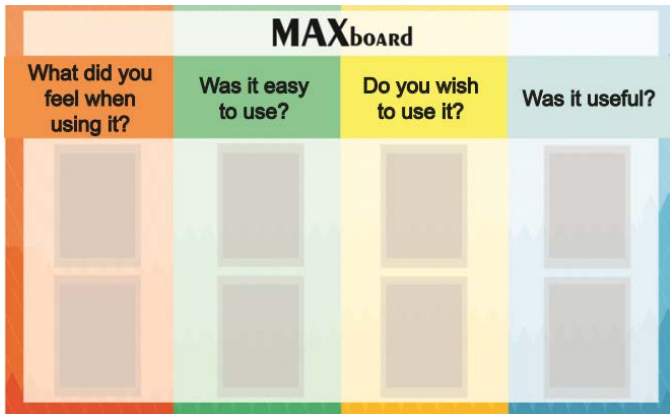


Figure 3. MAX Board (v1)

The application process of the MAX method is illustrated in Fig. 4. First the user must experience the software by carrying out tasks using a mockup, prototype of finished application (see Fig. 4 stage A). Then, the moderator presents the MAX method and the user must choose and place the cards in the board in order to report his/her experience (see Fig. 4 stage B). In this stage, the user is also allowed to express his/her opinions orally, explaining why (s)he chose a specific card. Finally, the evaluator collects and records the user's choice of cards and checks the occurrences of the cards associated with the reason for choosing them. It is important that the user is encouraged to talk about the choice of card as this information will help the evaluator to recognize what affects the user experience (see Fig. 4 stage C). Finally, the UX evaluator must generate a report listing the selected cards or if (s)he wishes, a picture of the board can be taken as a record of the evaluation.

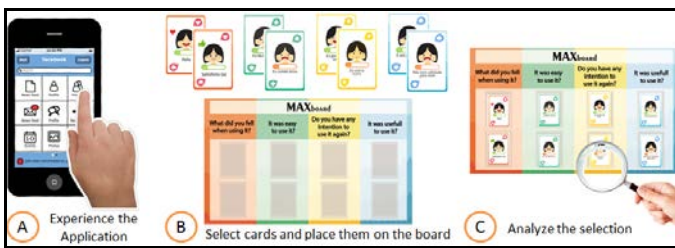


Figure 4. Application process of the MAX method: (a) Experience the Application; (b) Select cards and place them on the board; and (c) Analyze the selection.

IV. PILOT STUDY AND INITIAL IMPROVEMENTS ON THE MAX METHOD

As cited by Isomursu et al. [20], UX evaluation methods need to be to provide useful information to evaluators and positive experiences to users applying them. Thus, to evaluate the feasibility of the MAX method, verifying if it could be employed in the post-use UX evaluation of a system by users, we carried out a pilot study. This study aimed at assessing the opinion of users regarding the use of the method, evaluating if they felt comfortable when reporting their experience.

We carried out this study with three potential users of a Web application for a telecommunications company (which is referred to as ALFA to retain anonymity). The users signed a consent form before starting the study. The consent form explained the goals of the study and its activities, the anonymity of the subjects' data, and that the study was safe as users would only have to experience an application and provide feedback. Users were free to (dis)agree with participating in the study. The study had three stages: (a) testing the system, in which the users tried the application; (b) UX evaluation, in which the users carried the post-use evaluation using the MAX method; and (c) feedback, in which the users answered a questionnaire regarding their opinion towards the MAX method. Before carrying out the study, we prepared the following materials: (a) a consent form (explained above); (b) the MAX cards and a sketch of the MAX board (see Section 3); (c) a scenario describing the goals that the user had to accomplish in the evaluated system; and (d) the feedback questionnaire.

In the execution of the study, we invited the users to participate in the evaluation of the Web application of the ALFA company. The users were chosen by convenience and they tried to carry out the following tasks: access a detailed bill and print it. Then, the users employed the MAX method. We highlight that there was no need for training any software engineer in using the method. Since we aimed at evaluating MAX from the point of view of users, it was more suitable that an experience evaluator (one of the authors of the method) tested it with the users. Also, the users knew that they were able to end the evaluation session whenever they wanted.

At the end of the tests we took pictures of the chosen cards. Fig. 5 shows two different selections from two different types of users. The first selection of cards (Part A) was positive due to the experience of the user with the application. This user stated that (s)he was a client of ALFA and therefore, knew how to use the application. As (s)he managed to carry out the tasks, this user had a positive experience, indicating that (s)he felt satisfied and interested. Also, this user indicated that the application was intuitive and easy to use, and that (s)he would to use it again, because (s)he liked it. However, users experiencing the application for the first time did not have a positive experience. During the application of the MAX method, we identified the following problems regarding the system according to the users' reports: (a) the users had difficulty in requesting a password due to the unclear solicitation process; (b) the system feedback took time; and (c) the users had difficulties in finding the detailed bill, as they only found the paying value with no details on their service consumption. These problems negatively affected the UX of the subject whose cards selection is shown in Fig. 5 Part B. As we can see, this user felt satisfied but confused. Also, (s)he stated that the application was easy to use and intuitive, but not that much. And most importantly, that although the system could be useful and helpful, (s)he would not use it, or would use another system if available.

To assess the opinion of users towards the MAX method, users were asked to fill out a feedback questionnaire with the following questions: (a) *Does the method feel intuitive and easy to use?* and (b) *Is the method and the instruments related*

to it easy to use, learn and understand?. These questions were considered, as they have been employed by other authors evaluating the ease of use of UX evaluation methods from the point of view of user [20]. Also, we asked the subjects to indicate if they would use MAX again. Overall, the answers to the questionnaires were positive. For instance, the users indicated that the board helped them when selecting the cards due to its guide questions. Also, the users stated that the MAX cards were useful since the avatar was clear and they managed to associate the depicted emotion with the label on the card.



Figure 5. Selection of the MAX cards: (A) Positive and (B) Negative.

When describing their difficulties with the method, the users reported the high number of cards. Also, another user indicated that neutral cards were missing. This user stated that neutral cards were necessary because the first version of the MAX method only provides positive and negative cards for evaluating UX. Furthermore, in order to motivate the users to describe specific missing cards for depicting their opinion towards the evaluated application, we asked the following question: *“If I gave you a blank card to express any emotion/feeling you think is missing, would you suggest any?”* As a result, one of the subjects indicated the *“impatient”* card.

In general, the users indicated that they enjoyed employing MAX to describe their emotions. However, we still needed to improve the MAX method, based on the suggestions by the users. Regarding the cards, through our observations we noticed that the *“empathic”* card was not easy to understand and needed to be removed, as the users did not understand how they could feel empathy with an application. Also, we added the *“impatient”* card in the Emotion category and the *“I gave up the task”* in the Ease to Use category due to the users’ suggestions. Furthermore, since all users reported that there were too many cards, we reduced the scale from four to three intensity levels ending up with 75 cards (17 less cards than in the first version). Fig. 6 shows the changes over the second version (v2) of the MAX cards.

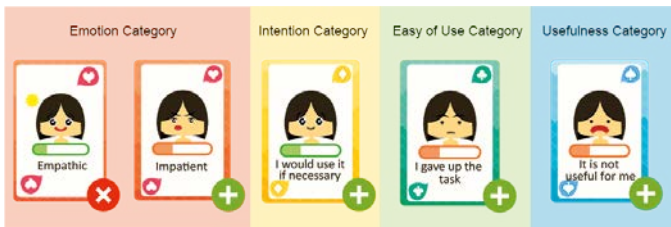


Figure 6. Changes over the MAX method (v2)

Regarding neutral cards, we chose not to add cards such as *“I did not feel anything”* or *“Indifferent”*, as users who choose these items do not provide information regarding the side to which they are inclined (either positive or negative UX).

Furthermore, regarding the board, we removed the slots. Although we intended to encourage users to choose two or more cards, the slots made users select only two cards. Therefore, besides providing a larger space for selecting the cards in each category on the board, we also reduced the size of the cards, so two or more cards could be selected. The new size of the cards was also motivated by ergonomic principals to facilitate its use by both evaluators and users. We also made design changes in the cards to make them more minimalistic.

V. CONCLUSIONS AND FUTURE WORK

This paper presented MAX, which intends to be an easy to use method for both evaluators and users. Since the board guides users through the evaluation process, the evaluators do not need to be experts, making this method suitable for software engineers willing to gain insight in terms of UX. Also, by providing cards and a board that are intuitive and informal, we aimed at making users feel comfortable so they would not feel that the evaluation was a chore, but a pleasant and informal activity to identify their needs.

During the pilot study, the MAX method was perceived as a useful tool that can easily and quickly capture the overall opinion of the users regarding the evaluated artifact. As the EmoCards [17], by providing different cards, the MAX method allows users to think of different possibilities, while evaluating further aspects such as ease of use, usefulness and intention to use. Also, when compared to scales, users can feel free to complement the reasons that made them choose a specific card, providing evaluators with information on UX problems that need to be corrected. Moreover, the MAX method allows evaluators to interpret the emotion/expression in the drawings more easily, as it provides labels allowing the understanding of the cards meanings.

One of the limitations of our study was the small sample size employed. However, even with the small sample the results from this pilot study allowed us to test the applicability of the MAX method with positive results. Also, as the subjects who participated in the study were real end-users, their problems when employing the method could affect future applications of the method. Therefore, even though we could wait to refine the method after further studies, we decided to include the changes described in section IV so we can test their impact in the future applications of the MAX method.

Another limitation is that this initial evaluation was performed from the point of view of users. Since the evaluator was one of the authors of the technique, the application process could have been easier, affecting the overall opinion of the subjects. Therefore, we still need to carry out another evaluation from the point of view of software engineers, to verify what is needed to facilitate the use of the MAX method and the analysis of the results from the evaluation. Also, we need to verify if the opinion of users towards MAX remains the same, when being applied by software engineers. However, we can argue that an initial evaluation from the point of view of users is necessary before carrying out evaluations in industry, which are more expensive.

There could have also been a threat to the validity of our results in terms of the evaluated system and its

representativeness. Although a Web application of a telecommunications company may not be representative of all types of applications (i.e. Web, mobile, desktop, others), it is still a real application which can yield different experiences. This study showed that MAX managed to capture such difference, by presenting opposing experiences from different users. However, we still need to verify the suitability of using MAX in evaluating other types of artifacts, such as prototypes, in which only part of the application has been developed. Finally, one last limitation is the instrument and measures applied in this study for assessing the users' opinion towards the MAX method. However, we believe that applying questionnaires was more suitable than applying interviews due to time constraints. Furthermore, evaluating if a UX technique is easy to use and to apply from the point of view of users is relevant in order to meet users' needs and make them feel comfortable during the evaluation process [7][20].

As future work we intend to carry out further empirical evaluations with the improved version of the MAX method in industrial and not controlled scenarios with more subjects/evaluators. Such evaluations aim at generalizing our results to further contexts and at evaluating if the current changes have an impact in the results of the UX evaluation. We also intend to carry out further comparative studies with the method described in Section II and other UX evaluation methods, to identify the situations in which MAX is more suitable to be applied. Among the improvement opportunities, we intend to develop other avatars for the MAX cards (i.e. with different genders and races) and evaluate their impact on the users' choice through comparative studies. Furthermore, while we intended to understand the degree of impact of an item from the MAX cards by its intensity, we noticed that the number of cards could have a cognitive overload over the users. Thus, we will check whether the scale also impacts the description of the UX through more empirical studies, and if needed, we will reduce it. Finally, we intend to develop a ready to use version of MAX, containing the instructions, cards and board. Such version will be useful for practitioners aiming at employing MAX at work or in the field.

ACKNOWLEDGMENTS

We thank CNPq for the scholarship granted to the second author of this paper and for its financial support through process n° 460627/2014-7. Also, we thank the financial support granted by FAPEAM through processes n°: 01135/2011; 062.00146/2012; 062.00600/2014; 062.00578/2014; and PAPE 004/2015. Finally, we would like to acknowledge the support granted by FPF Tech.

REFERENCES

- [1] J. Bargas-Avila, K. Hornbæk, "Old wine in new bottles or novel challenges: a critical analysis of empirical studies of user experience", In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2689-2698, 2011.
- [2] A. Fernandez, E. Insfran and S. Abrahao, "Usability evaluation methods for the Web: A systematic mapping study", Information and Software Technology, 53(8), pp. 789-817, 2011.
- [3] M. Hassenzahl, S. Diefenbach, A. Göritz, "Needs, affect, and interactive products—Facets of user experience", *Interacting with computers*, pp. 353-362, 2010.
- [4] International Standardization Organization, ISO 9241-210: Ergonomics of human system interaction, Part 210: Human-centred design for interactive systems, 2010.
- [5] E. Law, V. Roto, M. Hassenzahl, A. Vermeeren, J. Kort, "Understanding, scoping and defining user experience: a survey approach", Proceedings of Human Factors in Computing Systems conference, CHI'09, pp. 719-728, 2009.
- [6] M. Tähti, L. Arhippainen, "A Proposal of collecting Emotions and Experiences", In *Interactive Experiences in HCI*, Volume 2, pp. 195-198, 2004.
- [7] A. Vermeeren, L. Law, V. Roto, M. Obrist, J. Hoonhout, K. Väänänen-Vainio-Mattila, "User experience evaluation methods: current state and development needs", In Proceedings of the 6th Nordic Conference on HCI, pp. 521-530, 2010.
- [8] R. Miles, J. Greensmith, H. Schnadelbach, J. Garibaldi, "Towards a method of identifying the causes of poor user experience on websites", In *Computational Intelligence, UKCI*, pp. 258-265, 2013.
- [9] J. Preece, Y. Rogers, H. Sharp, "Interaction design: Beyond Human-Computer Interaction", Wiley New York, 2002.
- [10] V. Roto, "User Experience from Product Creation Perspective. Towards a UX Manifesto workshop", In *Conjunction with HCI*, pp. 11-15, 2007.
- [11] M. Hassenzahl, "User Experience – a Research Agenda", In: *Behaviour and Information Technology*, 25(2), pp. 91-97, 2006.
- [12] A. Moreno, A. Seffah, R. Capilla, M. Sanchez-Segura, M. "HCI Practices for Building Usable Software", *Computer*, pp. 100-102, 2013.
- [13] L. Yong, "User experience evaluation methods for mobile devices", In Proceedings III International Conference on Innovative Computing Technology, pp. 281-286, 2013.
- [14] P. Lang, "Behavioral Treatment and Bio-behavioral Assessment Computer Applications", In *Technology in Mental Health Care Delivery Systems*, pp. 119-137, 1980.
- [15] V. Venkatesh, M. Morris, G. Davis, F. Davis, "User acceptance of information technology: Toward a unified view", *MIS quarterly*, pp. 425-478, 2003.
- [16] M. Hassenzahl, M. Burmester, F. Koller, "AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität", In *Mensch & Computer 2003*, pp. 187-196, 2003.
- [17] P. Desmet, C. Overbeeke, S. Tax, "Designing Products with Added Emotional Value: Development and Application of an Approach for Research through Design", *The Design Journal*, pp. 32-47, 2001.
- [18] M. Hassenzahl, "The thing and I: understanding the relationship between user and product", In *Funology*, Springer Netherlands, pp. 31-42, 2005.
- [19] R. Plutchik, "The Nature of Emotions Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice", *American Scientist*, 89(4), pp. 344-350, 2001.
- [20] M. Isomursu, M. Tähti, S. Väinämö, K. Kuutti, "Experimental evaluation of five methods for collecting emotions in field settings with mobile applications", *International Journal of Human-Computer Studies*, 65(4), pp. 404-418, 2007.

Designing Personas with Empathy Map

Bruna Ferreira, Williamson Silva, Edson Oliveira, Tayana Conte

USES Research Group, Instituto de Computação - IComp

Universidade Federal do Amazonas (UFAM)

Manaus, AM - Brazil

{bmf, williamson.silva, edson.cesar, tayana}@icomp.ufam.edu.br

Abstract— A software product’s acceptance depends on the user experience that it provides to its users. The software product must meet the user needs, and one way to understand those needs is through creation of Personas. The Personas technique allows describing the users’ characteristics, goals and skills. The Empathy Map (EM) method can be used to describe personas. The EM’s goal is to create a degree of empathy with the user so the product developing team starts to understand more deeply the users and become more aware of their real needs. To assess the EM effects on the creation of personas, we conducted a feasibility study with 20 subjects. Initially, the subjects learned how to describe personas in textual form. After that, they applied the EM to create personas. After using the EM, the subjects answered a questionnaire about their perceptions regarding the EM’s ease of use and its usefulness. The results showed that majority of subjects considered EM useful and easy to create personas. Furthermore, this majority also said that they would use the EM for the creation of personas again.

Keywords- *Persona; Empathy Map; User Experience; UX.*

I. INTRODUCTION

The software development focuses on users’ needs and emotions while interacting with the product is critical for the software product success [1]. According to Sproll *et al.* [1], as the field of User Experience (UX) explores these needs and their fulfillment, it gains in importance against the background of the wish for human-oriented products and services. In order to develop usable systems is necessary to understand the users that will interact with the system [2].

One technique that can be used to better understand the users’ needs is the Personas technique. The Personas technique provides an understanding of the system user in terms of his or her characteristics, needs and goals to be able to design and implement a usable system [3]. The user modelling technique known as personas has obtained excellent results over the last years [4]. Furthermore, the Personas technique gathers data about users, gains and understanding of their characteristics, defines fictitious users (called personas) based on this understanding and focuses on these personas throughout the software development process [3]. Through the collected data using the Personas technique we can obtain greater knowledge of the user for which we are designing.

However, the creation of personas involves much creativity [5]. It is also difficult to verify if a persona really reflects user’s data [5]. The Persona technique is used in order to aid designers to create empathy with the users and identify users’ characteristics [2]. Empathy has been employed as a defining

characteristic of designer-user relationships when design is concerned with user experience entails [6]. Furthermore, to guide designers to describe personas, we adopted the Empathy Map (EM). The EM is a method that helps designing business models based on the client perspectives [7]. The EM template has a visual organization. This organization simplifies the template implementation. Furthermore, the EM has guide questions [7]. This guide questions aid the designers during creation of personas, making this process more systematically.

This paper presents the results of a feasibility study where the EM is employed for the creation of personas. In this study, we evaluated the perception of the subjects regarding to ease of use and usefulness of the EM for the creation of personas. Through the analysis of the results it was possible to obtain the user’s perception regarding the use of EM. In addition we identified improvement suggestions for the Empathy Map. The remainder of the paper is organized as follows. Section II presents the User Experience, Personas and Empathy Map concepts. Section III details the feasibility study, followed by our results in Section IV. The Section V shows the validity threats of the feasibility study. Finally, conclusions and comments on future work are given in Section VI.

II. BACKGROUND

A. User Experience

According to the ISO 9241 [8], User eXperience is defined as: “a person’s perceptions and responses that result from the use and/or anticipated use of a product, system or service”. The user experience explores how a person feels about using a product, i.e., the experiential, effective, meaningful and valuable aspects of product use [9]. The focus on the user’s needs and emotions while interacting with a product is a key factor for the product success [1]. Therefore, user experience modeling is especially important for understanding, predicting and reasoning about UX processes, with implications for the software design [10]. One way to understand the user’s needs is through the use of Personas.

B. Personas

Persona is a hypothetical archetype of a real user [12]. It describes the user’s goals, skills and interests [12]. In order to describe personas, it is important to detail their characteristics, such as: name, image, occupation, family, friends and age [11]. Designers can choose various ways to represent personas, but they are usually represented in textual form, enriched by a photo. Among the benefits of using Personas, Cooper [12] cites: (1) it helps the development team to understand the

characteristics of a group of users; (2) it proposes solutions related to the main users' needs; (3) it provides a human face to bring potential users closer to the team. The Persona technique is mainly criticized for being grounded in informal and unscientific data, for being difficult to implement, for not describing real people, and for preventing designers from contacting real users [13]. In summary, the usefulness and ease of use of the technique are often questioned.

C. Empathy Map

Empathy Map (EM) is a method that assists designing business models according to customer perspectives. It goes beyond demographic characteristics and develops a better understanding of the customer's environment, behavior, aspirations and concerns [7]. The EM's goal is to create a degree of empathy for a specific person [14]. According to Bratsberg [15], the EM is a user-centered approach, i.e., the focus is on understanding the other individual by looking at the world through his or her eyes. When the stakeholders understand the user, they are able to understand how small changes in design can have a big impact on users [15].

In the first version of the EM, Matthews [16], proposed four different areas that should be covered when making an Empathy Map of a person (see Fig. 1). After, Bland [16] improved the EM by including Pain and Gain areas. As a result, the EM consists of six areas: (a) See – what the user sees in his/her environment; (b) Say and Do – what the user says and how s/he behaves in public; (c) Think and Feel– what happens in the user's mind; (d) Hear –how the environment influences the user; (e) Pain– the frustrations, pitfalls and risks that the user experiences, and (f) Gain –what the user really wants and what can be done to achieve his/her goals. The EM also has a set of questions that guides how to fill the fields.

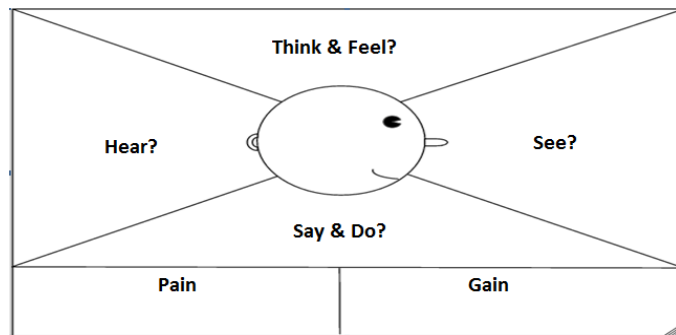


Figure 1. The Template of the Empathy Map [7].

III. FEASIBILITY STUDY

In order to verify the subject's opinion regarding the acceptance of the EM to create personas, we conducted a feasibility study with 20 volunteers' undergraduate and graduate students in Computer Science. In this study, the subjects should construct personas using both textual description and EM.

The subjects were attending a course on User Experience. All the 20 students agreed to participate in the feasibility study. We carried out the study in two days, during class time. In the first day, the subjects attended a class about the Personas technique. In order to create the personas, the subjects received

scenarios to extract the personas' characteristics. We employed two scenarios. These scenarios are related to an application to assist persons with epilepsy. The application was being developed and the scenario was created according the application requirements. The application has two users: (1) persons with epilepsy and (2) family of persons with epilepsy. The first scenario described the routine of a person who has epilepsy. The second scenario described a routine of a family member of a person who has epilepsy. The first scenario was used for the creation of Personas through the text description. The second scenario was used to create Personas using the EM.

On the textual template, the subjects had to describe the following Persona features: (1) description of who the persona is (name, age, profession, gender, and others); (2) information on the persona's housing (where s/he lives, who s/he lives with, and other housing features); (3) what problems the persona faces; and (4) the persona's expectations, i.e., what the persona found or needed that could help to solve his/her problems. Besides describing the features, the subjects had to draw the created persona.

In the second day, we presented the EM template and explained how to use it. Then the subjects extracted information of the second scenario to describe the persona. In that context, the employed EM template was composed of the following fields: (1) do; (2) feel; (3) think; (4) pains (difficulties/ frustrations) and (5) needs.

Such template does not have the same fields that the original template. To simplify the template, we pulled the fields: 'see', 'say' and 'hear' because these fields referred to features related to the environment that the persona lives and not related to the persona. The fields: 'think' and 'feel', that are presented together in the original template, were separated to make the subjects think about the "think" (thoughts and ideas) and "feel" (emotions) aspects that can influence the user experience. Besides filling the fields, the subjects had to draw the persona as in the previous method. Fig. 2 presents the template used by subjects for creating the personas.

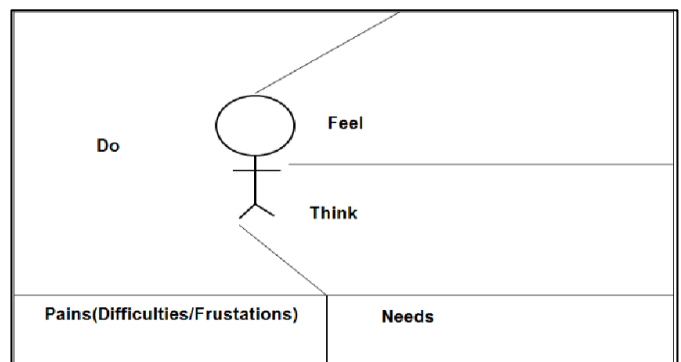


Figure 2. Empathy Map Template used in the study.

To fill the EM template, we provided some questions to help empathize with the persona; these questions are adapted from the original issues of EM. Each EM field had some specific questions. These questions are described in Table I.

After the personas creation, the subjects answered a questionnaire giving their opinions regarding the use of the EM

for the creation of personas. The subjects answered questions about the perceived ease of use and usefulness of the EM. Additionally, they answered questions regarding their intention of using the EM again and positives and negatives aspects of its application.

TABLE I. QUESTIONS FOR FILLING THE EMPATHY MAP [7]

Field	Guiding Questions
Do	What is common for him / her to say?
	How does s/he normally act?
	What are his / her hobbies?
	What does he like to say?
	How is the world in which s/he lives?
	What do people around him / her do?
	Who are his / her friends?
	What is popular in his daily life?
	What people and ideas influence him / her?
	What do the important people in his / her life say?
	What are his / her favorite brands?
	Who are his / her idols?
Think	What are some important ideas that s/he thinks and does not say?
Feel	How does s/he feel about life?
	What bothers him / her lately? Why?
Pains (Difficulties / Frustrations)	What is s/he afraid of?
	What are his / her frustrations?
	What has disturbed him?
	What would s/he like to change in his / her life?
Needs	What does s/he need to feel better?
	What is success? What does s/he want to achieve?
	What has s/he done to be happy?
	What would end his / her pain?
	What are some of his / her dreams?

In this study, we used factors defined within the Technology Acceptance Model (TAM), such as ease of use and usefulness [17] to investigate the subject's acceptance regarding the EM applied in the creation of personas. The TAM model is based on two factors [18]: Perceived Usefulness and Perceived Ease of use. On the questionnaire we employed a six points scale with the items: totally agree, strongly agree, partially agree, partially disagree, strongly disagree and totally disagree. We did not use an intermediate level as suggested by Laitenberger and Dreyer [18] since this neutral level does not provide information regarding the side to which the subjects are inclined (either positive or negative). In this questionnaire, the subjects answered a set of questions that measure the perceived usefulness and ease of use.

Besides the questions to be answered, we added three questions to the questionnaire to obtain more feedback about the subjects' perception regarding EM. The questions added to the questionnaire are presented in Table II.

TABLE II. SUBJECTIVE QUESTIONS ADDED IN THE QUESTIONNAIRE

Nº	Question
1	If you had to use personas again, would you choose the traditional way or the Empathy Map? Why?
2	What aspects of the Empathy Map do you consider positive for the creation of personas?
3	What aspects of the Empathy Map do you consider negative for the creation of personas?

IV. RESULTS

In this section, we describe the analysis of created personas generated by both methods. Furthermore, we describe the results regarding to the obtained answers from the subjects to the questionnaire.

A. Perception about the Empathy Map's Usefulness

Fig.3 shows the answers to each statement related to the perceived usefulness of the EM. Table III shows the factors evaluated in the perceived usefulness of EM.

TABLE III. STATEMENTS OF THE USEFULNESS

ID	Statements
U1	Using EM would enable me to create Personas more quickly.
U2	Using EM would improve my performance when creating personas.
U3	Using EM would increase my productivity when creating personas.
U4	Using EM would enhance my effectiveness when creating personas.
U5	Using EM would make it easier to create personas.
U6	Using EM would be useful for creating personas in my projects.

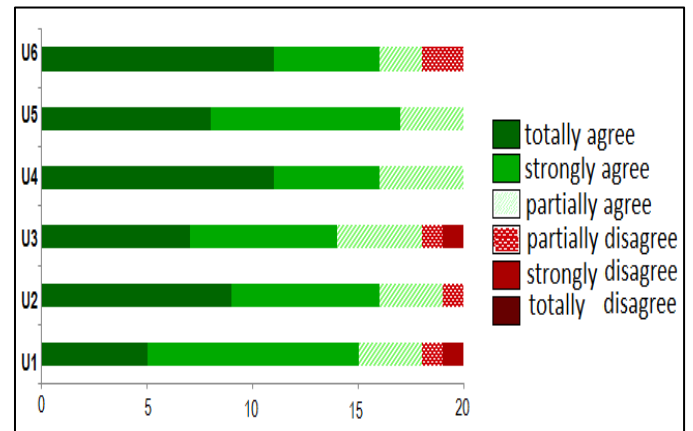


Figure 3. Results Regarding the Perceived Usefulness of EM

Regarding how quickly it was to create Personas using Empathy Map, only 02 out of 20 subjects disagreed that the EM helps creating Personas more quickly (U1). Regarding improved effectiveness on the creation of Personas (U4), i.e., to better describe the Persona using EM, no subject disagreed.

Regarding the performance in the creation of Personas (U2), i.e., being able to better characterize the persona using EM, only one subject disagreed. The subject that disagreed stated that the guiding questions were difficult to understand. Perhaps the difficulty in understanding had influenced subject's performance. All the 20 subjects agreed that the Empathy Map facilitated the creation of Personas (U5). Moreover, regarding productivity increase in the creation of Personas (U3), only two subjects disagreed. Finally, of the 20 subjects, 18 agreed that EM would be useful for creating Personas in their projects (U6). The results regarding usefulness showed that most of the subjects considered the EM useful for creating Personas.

B. Perception about the Empathy Map's Ease of Use

Fig.4 shows the answers regarding the perceived ease of use of the EM. Table IV shows the factors evaluated in the perceived ease of use of EM.

TABLE IV. STATEMENTS OF THE EASE OF USE

ID	Statements
E1	Learning how to works the EM would be easy for me.
E2	I understood what I had to provide in every part of the EM.
E3	It is easy to remember how to create personas using EM.
E4	Using EM it was easy to create the persona that I wanted.
E5	It was easy to become skillful in creating personas using EM.
E6	I find EM easy to use.

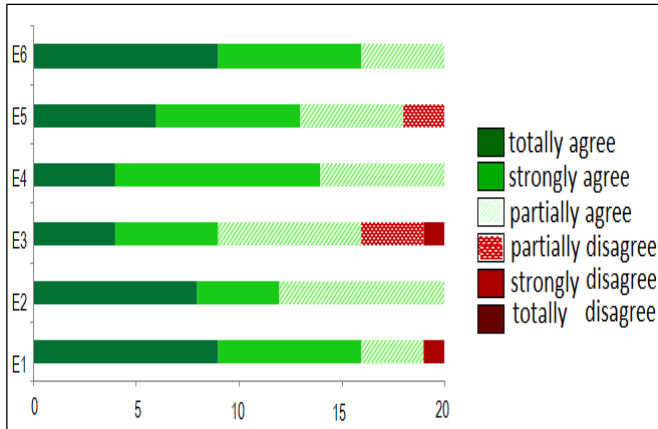


Figure 4. Results Regarding the Perceived Ease of Use of EM

Regarding the ease of learning to use the EM (E1), only one subject disagreed. All the subjects agreed that they were able to use the EM to create Personas as they wanted (E4). Regarding the understanding of the EM fields (E2), 4 out of the 20 subjects disagreed.

The difficulties in understanding, as well as other problems in the EM use will be discussed in the next subsection. All the subjects agreed that it was easy to gain the ability to use the EM (E5). From the 20 subjects, only 2 disagreed that is was easy to remember how to create Personas using EM (E3). All the subjects agreed that the Empathy Map was easy to use (E6).

C. Qualitative Results

Other way to investigate the point of view of subjects is to use qualitative methods. The use of qualitative methods allows the researcher to consider human behavior and thoroughly understand the studied object [19]. The qualitative analysis performed in this work is based on procedures from the Grounded Theory (GT) method. Grounded Theory is based on the coding idea that is the process of analyzing the data [20]. The Table V presents the results of the qualitative analysis.

TABLE V. RESULTS OF THE QUALITATIVE ANALYSIS

Category	Cotations
easiness in the description of personas	EM is more flexible than the traditional approach. <i>'The Empathy Map (...) facilitates, by providing the idea of almost being a defined guide, but it is flexible and you can add whatever you want in order to complete the description of the persona.'</i> – Subject 16
	EM guides inexperienced designers. <i>'(...) I think that it (EM) can certainly be an initial step for anyone who is learning to identify personas.'</i> – Subject 20

Category	Cotations
	The EM's fields guide the creation of personas. <i>'The highly detailed description of the persona, the way s/he acts, thinks, and his / her fears ... I believe that categories help describe the personas.'</i> – Subject 12
	EM deals with the subjective aspects of a persona. <i>'...it captures what the user 'feels' and 'thinks' which I did not see in the traditional approach.'</i> – Subject 18
difficulties in understanding the EM	Difficult to answer the guiding questions. <i>'...it is difficult to answer the questions used as a guide, creating some uncertainty over where certain descriptions fit, i.e., which would be the correct quadrant'</i> – Subject 8
	Questions seem to be similar for different fields on the EM. <i>'The questions seem similar in some categories and can confuse at the moment of filling them.'</i> – Subject 3
	Different fields of the EM appear to have the same meaning. <i>'...the Empathy Map seems to confuse in some parts that need to be filled. For instance, 'feel' and 'pain' seem to be redundant'</i> – Subject 5
	Confusion regarding on which field to fill in some information (which generates duplicated information in the persona). <i>'Sections 'needs' and 'pain' are very similar to the section 'What do you think', which can generate duplicated content'</i> – Subject 20
limitations	The scenario influences the completeness of the persona. <i>'The completeness of the persona also depends on the data available on the lifestyle, habits, among others.'</i> – Subject 11
	The structure of EM only helps if you have questions guide. <i>'Although the aspects of the map are clear (through the words that define them), they leave each aspect much broader. Without questions the answers (to fill in the map) would certainly be very vague.'</i> -Subject 4
improvement suggestions for the EM	Context field missing in the EM. <i>'The lack of a context field <background>'</i> – Subject 19
	It should create a relationship between the personas. <i>'(...) in the case of personas that relate to others, there could be an identified relationship with the other personas'</i> – Subject 13
	Guiding questions should be incorporated in the EM. <i>'(...) questions like a model could accompany the process of creating the persona'</i> – Subject 16
	More space for filling the fields in. <i>'I believe that the template could be optimized, providing larger space for some topics.'</i> – Subject 12

In this subsection, we observed that the qualitative research helped us identify categories and relationships of factors that influence the use of the Empathy Map.

V. VALIDITY THREATS

Every study possesses threats that can affect the validity of their results [21]. This subsection presents the threats to validity considered in this feasibility study. The textual form to create personas and the EM method had equivalent training. However, the results obtained through these methods cannot be directly compared because the scenarios used to create the personas were different. The scenario used to create the persona using the textual form was simple and it gave more details to persona creation. It was a basic scenario, in order to introduce the concept of personas to the subjects. Differently, the scenario to create personas using the EM was more elaborated. Furthermore, the textual form was used before the EM. This may have caused a learning effect. However, in this methodological approach, the subjects should understand the basic way to create personas before using the EM. Additionally, the level of education and knowledge of the subjects is also a validity threat.

VI. CONCLUSIONS

This paper presented a feasibility study that aimed at verifying the subject's acceptance of the EM when employed in the creation of Personas. Based on the study's quantitative results, we perceived that most subjects think that the EM is easy to use and useful for the creation of personas. Through the qualitative analysis, we identified some features that are directly related to the use of the EM in the creation of personas. One of the results of the qualitative analysis showed that through the EM it is easy to describe personas. One of the reasons is that the EM provides more flexibility than the textual description. It also guides inexperienced practitioners through the creation process. We also observed that the guiding questions help subjects to fill the EM. We also found some limitations in the use of the EM for the creation of personas. Additionally, through the qualitative results, we identified some improvement suggestions for the EM.

According to results obtained from the qualitative and quantitative analysis, we observed that the EM method had a good acceptance. This method was considered easy to use and useful for the most of the subjects. Therefore, the results indicated that the EM is a good method to help the process of personas creation. The improvements identified on the qualitative results served as basis to we improve the EM template and make the method better to software engineer's use. Furthermore, we will also carry out a study in the industry. In such study, the EM will be employed by software engineers to help them design an application.

ACKNOWLEDGMENT

We thank all the students who participated in the feasibility study. And we would like to acknowledge the financial support granted by FAPEAM through processes numbers: 062.00146/2012; 062.00600/2014; 062.00578/2014; 01135/2011 and PAPE 004/2015.

REFERENCES

- [1] S. Sproll, M. Peissner, C. Sturm, "From product concept to user experience: exploring UX potentials at early product stages" in 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries. ACM, 2010. pp. 473-482, 2010.
- [2] S. T. Acuña, J.W. Castro, N. Juristo, "A HCI technique for improving requirements elicitation," in Information and Software Technology, v. 54, n. 12, pp. 1357-1375, 2012.
- [3] J. W. Castro, S. T. Acuña, N. Juristo, "Enriching requirements analysis with the personas technique," Proceedings of the Intl. Workshop on: Interplay between Usability Evaluation and Software Development (I-USED 2008). pp. 13-18, 2008.
- [4] T. Ribeiro, P. Souza, "A Study on the use of personas as an usability evaluation method," in 16th International Conference on Enterprise Information Systems (ICEIS 2014), pp. 168-175, 2014.
- [5] T. Matthews, T. Judge, S. Whittaker, "How do designers and user experience professionals actually perceive and use personas?," in Conf. on Human Factors in Computing Systems, pp. 1219-1228, 2012.
- [6] P. Wright, J. Mccarthy, "Empathy and experience in HCI". in Conf. on Human Factors in Computing Systems. ACM.. pp. 637-646, 2008.
- [7] A. Osterwalder, Y. Pigneur, "Business Model Generation", Alta Books, 2013.
- [8] ISO 9241-210:2010. International Standardization Organization (ISO). Ergonomics of human system interaction -Part 210: Human-centred design for interactive systems. Switzerland, 2010.
- [9] P. Vermeeren, C. Law, V. Roto, M. Obrist, J. Hoonhout & K. Väänänen-Vainio-Mattila. "User experience evaluation methods: current state and development needs," in 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries. ACM.. pp. 521-530, 2010.
- [10] E. L. C. Law, S. Abrahão, A. P. Vermeeren, E.T. Hvannberg, "Interplay between user experience evaluation and system development: state of the art," in Int. Workshop on the Interplay between User Experience (UX) Evaluation and System Development (I-UxSED 2012), pp. 1-3, 2012.
- [11] J. Grudin, J. Pruitt, "Personas, participatory design and product development: An infrastructure for engagement," in: PDC. 2002. pp. 144-152, 2002
- [12] A. Cooper, "The inmates are running the asylum: Why high-tech products drive us crazy and how to restore the sanity," in Sams Publishers, 1999.
- [13] L. Nielsen, K. S. Nielsen, J. Stage, J. Billestrup. 'Going global with personas.' International Conference on Human-Computer Interaction, INTERACT 2013. Springer Berlin Heidelberg, pp. 350-357, 2013
- [14] D. Gray, S. Brown, J. Macanuso, "Gamestorming – A playbook for innovators, rulebreakers and changemakers," in Sebastopol, CA: O'Reilly Media, Inc., 2010.
- [15] H. M. Bratsberg, "Empathy Maps of the FourSight Preferences," in Creative Studies Graduate Student Master's Project, Buffalo State College. Paper 176, 2012.
- [16] D. Bland, "Agile coaching tip—What is an empathy map?," Available in <http://www.bigvisible.com/2012/06/what-is-an-empathy-map/>, 2012.
- [17] B. Langefors, "Discussion of the Article by Bostrom and Heinen: MIS Problems and Failures: A Socio-Technical Perspective. Part I: The Causes [MIS Quarterly, September 1977]. 1978.
- [18] O. Laitenberger, H. M. Dreyer, "Evaluating the usefulness and the ease of use of a web-based section data collection tool," In 5th International Symposium on Software Metrics, pp.122-132, 1988.
- [19] C. B. Seaman, "Qualitative Methods". In: Guide to Advanced Empirical Software Engineering (Shull *et al.*. (eds.): Springer., pp. 35 – 62, 2008.
- [20] A. Strauss, J. Corbin, "Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory," in Thousand Oaks, CA, SAGE publications, 2014.
- [21] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Essl, "Experimentation in Software Engineering: An Introduction", Kluwer Academic Publishers, 2000.

An approach to identify relevant subjects for supporting the Learning Scheme creation task

Huander Tironi

Post Graduate Program in informatics
(PPGIA) – Polytechnic School
Pontifícia Universidade Católica do
Paraná - PUCPR
Curitiba, Brasil
huander.tironi@gmail.com

André Menolli

Computer Science Department
Universidade Estadual do Norte do
Paraná - UENP
Bandeirantes, Brazil
menolli@uenp.edu.br

Sheila Reinehr, Andreia Malucelli

Post Graduate Program in Informatics
(PPGIA) - Polytechnic School
Pontifícia Universidade Católica do
Paraná – PUCPR
Curitiba, Brazil
sheila.reinehr@pucpr.br,
malu@ppgia.com.br

Abstract—The necessity to improve performance of the processes within organizations, gave rise to many research that apply concepts from educational area in software development companies. Many studies are related to Organizational Learning (OL), an area that helps companies to improve their processes significantly through the reuse of experiences. In recent works, some approaches propose to generate courses in organizations from content produced by employees. The main limitation of these approaches is the high dependence of an expert, who is responsible by the courses. Even a qualified expert, can be unfamiliar with the real need of the team's learning, and mapping the organizational needs requires time and effort. This work presents a mechanism for software development companies, capable of recovery searches performed by employees on the internet, in order to discover the real necessity of the team's learning. From these needs is purposed a learning schema of a unit of learning (the structure of a course), so helping the expert in the course creation task. An initial experiment was conducted and the results indicate that the use of the approach is viable and may help an expert create units of learning, assisting to improve the OL in software development teams.

Keywords- *Organizational Learning; Unit of Learning; IMS Learning Design; Learning Scheme; Recommendation System*

I. INTRODUCTION

Nowadays, the impact of knowledge on organizations is so relevant that it is not treated as a strategic factor in potential, available to few privileged, but as a common element essential to the company survival [1]. Knowledge is vital to corporations, especially for intensive knowledge company. The intensive knowledge projects refer to those where most work is said to be of intellectual nature and qualified employees form the bulk of workforce [2].

On the software development, the technical expertise that each employee acquires with the business practices and routines is valuable for the organization. Thus, as time goes by, the experiences and lessons learned gained make the software professionals more valued, becoming them in a source of basic knowledge to the company. However, the high value given to these employees creates an interest by other companies on these professionals. Losing an experienced employee to another company, means losing the acquired knowledge over time [3].

This situation makes organizations look for ways to store and share the knowledge generated. The field that seeks minimize those problems is the Organizational Learning (OL), which deals with the capacity or processes within an organization to maintain or improve performance based on experience [4].

Some recent researches are applying concepts from educational area, such as Learning Objects (LO) and Units of Learning (UOL), to improve OL in software development companies. A LO is defined as any independent digital or non-digital entity that may be reused in several teaching contexts [5]. Furthermore, a UOL can be seen as a general name for a course, a workshop or a lesson that can be instantiated and reused many times by different people and in different settings in an online environment [6].

Based on this context, the work of Menolli, Reinehr and Malucelli [7] proposes a semantic collaborative environment for software development companies. The environment aims to organize the content generated using social tools in learning objects and later, using a learning design defined by an expert, create units of learning, using semantic technologies. However, in this approach, the creation of courses depends directly of an expert, who defines a course structure, using a Learning Scheme (LS). LS is a structure defined on a meta-language, e.g. XML, which the tags form a structure that contains elements from a course such as a process of teaching and learning [8].

This dependency makes indispensable the presence of an expert, who can assume a high cost position. However, even the expert taking a high position; he can be unfamiliar with the real need of the team learning. To map the team needs requires time and effort and can be a barrier to set a Learning Scheme.

Therefore, having exposed these limitations it is necessary to advance, regarding learning, and present an approach that assist to identify relevant subjects and content to the team.

Hence, this paper presents a mechanism for software development companies, capable of recovery searches performed on the internet by the employees, and then, using a clustering algorithm, to group this searches, helping on the Learning Scheme creation task.

The remaining parts of the paper are organized as follows: Section II presents background information on the main

concepts behind the proposed mechanism, such as Learning Scheme, Clustering, Text Mining and Recommendation Systems. Section III shows some related works. Section IV introduces the proposed mechanism as well as its architecture. Section V presents an experiment and Section VI presents the final considerations about the study.

II. BACKGROUND

A. Learning Design

The structure of a UOL is defined using some kind of Educational Modeling Language (EML), that are models of semantic information or aggregations, that describe, of a pedagogic point of view, a content as well as educational activities [9].

The EML are organized on units of study in order to allow its reuse and interoperability [10].

One of the main Educational Modeling Language is the IMS Learning Design (LD) [11], which supports the use of different approaches of teaching or learning, such as: behaviorists, cognitive and constructivist. The model describes “Units of Learning”, as elemental units that come learning events for learners, satisfying one or more learning objectives.

The IMS Learning Design specification is a meta-language that describes all the elements of the project of a process of teaching and learning, elaborated by the work group IMS/GLC [11]. The IMS LD describes a method comprising by a series of activities conducted by both the student and the team, in order to reach the learning objectives [9].

However, the IMS LD is much more complex than only organizing knowledge in a course form. Menolli, Reinehr and Malucelli [7], proposed an adaptation of IMS LD to become viable its use on an organizational learning environment. The main differences between the learning design proposed on Menolli, Reinehr and Malucelli [7] and the actual IMS LD, is that on the proposed environment the components related with time and execution control on a UOL were not used. As the purpose of this work is to present an advance on an approach already proposed, it is been used the same concepts.

In this approach, the UOL contains Resources and is organized as a Learning Design, which contains definitions such as Pre Requisites and Learning Objectives. The LD is also bound to an activity which contains a Description and its Structure.

This information is organized in a XML file called Learning Scheme. The Learning Schema follows the IMS LD structure, and contains information about the course, such as objective and prerequisite, beyond the activities of learning as well its hierarchy and sequence.

B. Recommendation Systems

There is an extensive class of Web Applications that involve predicting user responses to option. Such a facility is called a recommendation system [12].

There is a list of applications of recommendation systems that goes from Products to News Recommendation, but there is a few applications aimed to learning. Some [13] proposed a semantic recommendation system for e-learning domain to help

the learners find subject they need to learn based on learners knowledge level, learners profile and some learners evaluation. Also is presented [14] a model to improve proactive context-aware recommendations in e-Learning systems to be applied in online e-Learning authoring tools.

There are two basic architectures for a recommendation system: Content-based systems examine properties of the items recommended; and Collaborative filtering systems, that recommend items based on similarity measures between users and/or items [15].

However, the one which fits better to the proposed mechanism is called Content-Based. The Content-Based systems focus on properties of items. Similarity of items is determined by measuring the similarity in their properties [16].

In a content-based system is necessary to build each item a profile, which is a record of collection of records representing important characteristics of that item. In simple cases, the profile consists of some characteristics of the item that are easily discovered. But, there are other classes of items where it is not immediately apparent what the values of features should be [17]. It is considered to this work one of them: words or documents collections.

In order to identify these words, we proceed with some practices of text mining called Filtering, which is a list of words to discard because they represent low-semantic words (prepositions, etc), and Stemming words to achieve a canonical concept representation (e.g. analysis, analyzing, analyser are collapsed to ANALY).

Once the documents are represented by sets of words, is necessary to measure the similarity of two or more documents, and to that there are several natural distance measures can be used, such as Jaccard similarity coefficient [12] between the sets of words, or cosine distance between the sets, treated as vectors.

III. RELATED WORKS

In recent years, organizations have begun to place more value on the experience and know-how of their employees, i.e., their knowledge [18]. Therefore, it has become a challenge to develop and implement processes that generate, store, organize, disseminate and apply the knowledge produced and used in a company in such a way that it can be systematically and reliably accessed by the organizational community [7].

In recent years, software companies have used tools and technologies to knowledge management that were not designed for this specific purpose [19]. The arising of the Web 2.0 (blogs, wikis, content sharing sites, social networks, etc.) gives access to a growing need for Recommendation Systems based on social and information network mining methods [20].

More and more companies are interested towards the integration of Recommendation Systems in the Intranet in order to further improve communications [20] and organizational learning.

In the work of Reichling, Veith and Wulf [21] is proposed an expertise recommender system for the specific needs of a major European national industry association. Other studies have been

proposing knowledge management systems such as Luo and Cao [22] that presents an architecture to realize knowledge sharing and knowledge recommendation based on user model. Also is presented by Ale et al. [23], an architecture to provide a technological support for knowledge representation and retrieval activities.

The growing number of works related to organizational learning area shows that the search for techniques for improving learning in teams is recurring and current on software factories. The environment proposed in this study is an improvement of the consolidated approach developed by Menolli, Reinehr and Malucelli [7] and is presented in the section the follows.

IV. ENVIRONMENT

The proposed approach aims to generate the Learning Scheme using data from the searches performed by the employees on search engines, such as Google, Yahoo and Bing. The reason of choosing queries typed into search engines as source of information is because more and more, software developers are using search engines to find techniques, coding solved issues and new technologies [24]. According to QuantCast, the online programming forum “StackOverflow”, increased from around one thousand accesses to more than three million visits per day since 2009. It shows that many programmers get answers for their needs on the internet, submitting a question or searching search for answers to a question that has already been made.

This approach is divided into two components. The first is called “Themes and Roles Identifier” (TRI) and works as a collector of queries typed on search engines and user’s information. This component uses the collected information to suggest to the expert the themes and roles most searched.

The second component defines a course from all the information gathered on the TRI and defines the UOL structure. This component is called the “Course Definer” (CD). As a final task, based on the Learning Design, the CD will set a Learning Scheme, which finally can be used to create a Unit of Learning.

A. Themes and Roles Identifier

This component has as main objective to present suggestions of course’s themes and users that may participate or teach those courses. To do that, we collect the employee’s queries in search engines, e.g., “Java Polymorphism Examples”, and the employee’s information such as IP addresses, date and time of search. So, it is used text mining practices and a clustering algorithm to group these data and discover what have been the most searched queries by the group.

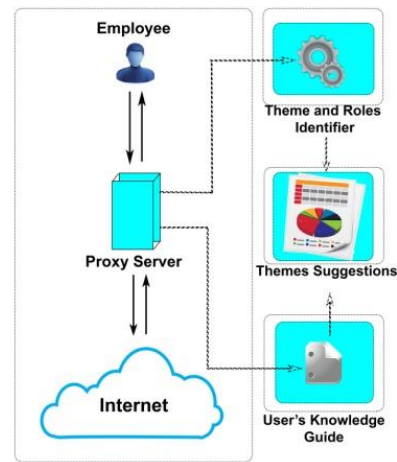


Figure 1. Approach to collect and generate the themes suggestions.

To obtain these topics is necessary to delineate a strategy to present a way of, by the analysis of a Proxy Server’s Log, to generate a search engine query report. The approach to collect both the theme and the employee’s information is presented on Figure 1.

Thus, with a proxy server mediating the connection between the user and the search engine is possible to capture the log of each query. However, an access log usually is not an easy reading file.

To get useful information from these records is needed to understand a record line. These records are divided into columns. First column refers to a Unix date and time, that after converted, becomes readable, for example:

1413858715.311 = Tue, 21 Oct 2014 02:31:55

The third column refers to the IP address that attempted to connect. Companies usually have a fixed address to all employees or computers, which enables pointing the employee to the connection he attempted. The proposed environment allows the expert to register the employees’ information such as name and skills, and then, bind this information to the IP addresses used by the employees in the company network.

The most important information for the proposed environment is the query. To collect this information was necessary to look into each record and learn what splits the typed query, from the rest of the record. As seen in most records, some HTTP parameters on a query request differ from the normal requests. It is crucial to get only queries arising from a search engine, such as Google, then, to split the query from the record we look for parameters that mark the beginning and the end of the queries. After splitting the query, all the gathered information is record on a database.

After recording data, the result that we have is a list of IP addresses, dates and queries searched by the employees. However, even that these queries represent a necessity of an employee, not all words typed into a search engine influences the meaning of the query. As the objective is to provide relevant

information to help creating a course, the solution to extract the important words was to use some techniques from text mining like tokenization, filtering and stemming.

After recording the important queries and its details, the next step is to cluster these records. The algorithm chosen to cluster the queries is the Cliques. The reason of using Cliques is because on the proposed environment, we seek cohesion between the elements of the same group, and the clustering algorithm provides it. The final result is a list of searches sorted by the most queried theme.

B. Course Definer

The aim of the Course Definer (DC) is to use all information obtained in the previous step, to help an expert to create a course.

The objects that are being treated in this component are texts and they represent a learning necessity of the group. It is an expert's work, to look onto the team's learning necessity and understand what the objectives and methods will be used to help improve the knowledge. Other concepts bound to a course structure still needs the choice of a professional, therefore, the proposed environment can facilitate this process and make it more interactive.

Information such as pre requisites and level of difficulty may be presented in the environment, using employee's information, as well as learners and staff levels. Hierarchy and contents may be suggested to the expert so that he can complete the course structure. The final task of the DC is to write the XML learning scheme file, which could be used by the expert to generate the course with its contents in an environment like proposed by Menolli, Reinehr and Malucelli [7].

C. Architecture

This section presents an architecture that gathers and organizes the components of the proposed environment, to create the learning schema.

Figure 2 provides a general overview of the proposed mechanism architecture. The architecture is subdivided into three tiers: application, middleware, and server log and database. The last block presented in this architecture, called Internet Connection, treats of the connections made by the employees and their searches.

The Application Tier is responsible for the user interaction and provides subsidy for the content inclusion, such as employees information, and creation of course structure. This tier is composed of four main structures that may be feed by the information collected and treated by the other tiers. The structures defined by the IMS Learning Design are: themes; roles; contents; and hierarchy.

The Middleware Tier provides a combination of Information Recovery, Text Mining and Clustering techniques. In addition, it makes, as a final task, the Course Structure Transformation, which takes all the data clustered and applies other text mining and clustering algorithm, but now, identifying the kind of course's structure, e.g., "Examples", "Concepts", "Exercises", etc. After rendering these data, the Middleware Tier presents them to the Application Tier, separating the information collected between the structures of the Learning

Design, such as "Theme", "Roles", "Contents" and "Hierarchy".

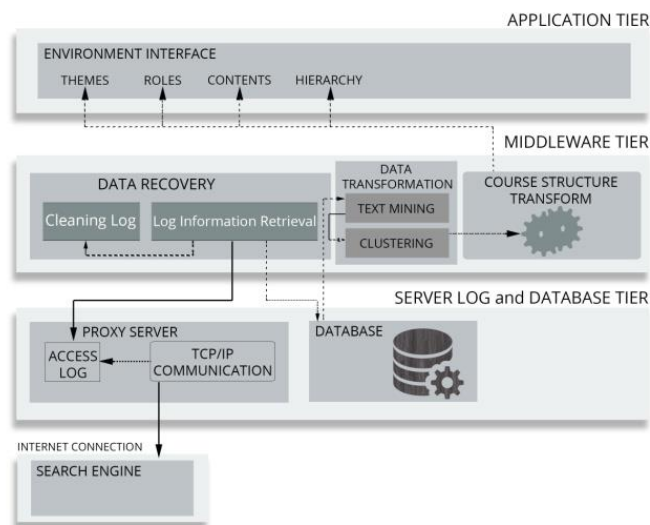


Figure 2. Proposed Architecture.

The Server Log and Database tier is responsible for interoperate between the user and the Internet Connection Tier, gathering and storing the connections attempts into a log file, and also storing the information found, after retrieving this from the server.

Thus, the objective of this architecture is the generation of courses structures, through the recovering of information shared by the team members on the internet search engines.

V. EXPERIMENT

The main objective of this experiment is to analyze if the suggested themes based on the queries performed by the development team can be considered relevant. To achieve this goal were elaborated some specific objectives that must be performed in this experiment:

- Present the environment to a development team;
- Validate the application of the approach in an corporative environment from the perspective of the employees;
- Analyze the suggested themes;

We proposed the use of a development team from a company of the vehicular tracking branch. In this experiment, four employees were selected, among which, all are linked to the development of software with a minimum of two years experience.

The areas of the software development included were interface and business layer programming and database management and analysis.

To manage the information about the employee responsible for a query, it was designed a screen which allows the expert to Create, Read, Update and Delete (CRUD) employees, as well as their specific knowledge, its levels, and the IP address used by

this employee on the network. To simulate an organizational environment, a local network was built. Also, it was created four employees with specific skills and its levels. The Employees were named E.1, E.2, E.3 and E.4. Each employee was mapped with an IP address into the local network, so that the mechanism could cross the search with the user.

After created the four employees, it was necessary to collect their searches on a search engine. Each employee was asked to perform at least fifty queries on the search engine. The criteria to perform the queries, was that the subject should be related to both the projects of the company and the developer skills.

Thereafter, the employees' searches were recovered from the proxy server log and stored on the database. Then, the Data Transformation component created from the stored queries, a list of bag of words. An algorithm of stop words removal was used to maintain only the words that represent the developer needs.

In order to present those searches grouped as themes, the Jaccard Similarity Coefficient was calculated to each pair of bag of words. Jaccard Coefficient uses the ratio of the intersecting set to the union set as the measure of similarity. Thus it equals to zero if there are no intersecting elements and equals to one if all elements intersect.

The average Jaccard Index established in previous simulations was 0.7, then, to this experiment, with the purpose of finding the best clusters, the same index was used.

After clustered, queries were presented as themes by the mechanism along with the amount of times that such searches were performed by the employees. Despite the tasks given to each employee were focused on different tasks, some queries performed by the users had the same theme. The main themes researches brought by the mechanism were about the interface framework called knockoutJS. Most of queries pointed to the specific words "knockout bind context", "nested foreach knockout" and "computed function knockout". Other queries pointed to themes related to the programming language C#, e.g. "C# MVC partial view", "C# trend line calc" and "Json serialize into object C#".

In order to validate the themes suggestions and the approach, a questionnaire was applied to the development team that participated in this experiment.

The results presented that beyond using the search engines to look for answers to their development needs, there is an incentive of the company managers on improving quality by allowing the use of search engines to that end. Also is shown that despite this approach of collect the internet log might seem intrusive, the development team considered important the fact of the company know the learning necessity of them.

Regarding the suggested themes, the developers were questioned about the necessity of learning on the topics that the mechanism presented as the most searched theme. The results presented as expected that the most searched themes were a group necessity. Also, the answers pointed that the words inside the themes were related to each other, showing that the mechanism didn't mix different queries in a single theme.

The experiment proceeded to the course definer component. The expert selected the themes suggested by the mechanism that

were related to the technology "knockoutJS", then the Course Definer crossed the results with the IP Addresses in order to map the employee responsible for each search. The employees pointed for the themes selected where E.1, E.2 and E.4. None of the searches performed by E.3 were present on the selected themes, because this specific developer was not involved on interface interaction, but only data processing.

With the themes selected, the Course Definer suggested a course where the main key words were the ones found in all the selected themes. In this case, the key word suggested was "knockout".

In order to suggest contents to the course, the course definer searched again in the queries to find other themes that may belong to the main theme. As the word "Knockout" represent a technology, this word appeared in other themes suggested, but combined with other words, such as: "css bind", "observable array". Thus, these themes were suggested along with the main theme, as contents to the course.

On the next step, the expert defined that to this course the main skill needed was KnockoutJS. To point and suggest the roles (learner and staff) into the course, the course definer looked into the employee's skills that were related to the main technology defined in the course, i.e., "KnockoutJS". Employee E.2 was the only having a senior level to the skill needed. Because of this, E.2 was pointed by the CD as the main tutor to the course. E.1 and E.4 were pointed as learners because their skill levels were plenum and junior respectively. The expert stills had the option of changing the roles of the employees and include new employees to the course. We choose to include the E.4 to participate of the course as a learner.

After that, in the Course Structure Transformation component, the expert was able to determine what kind of activities the course would have, such as: concepts, examples, advantages, disadvantages; and when to use the concepts learned. The activities chosen to the course created were concepts, examples, exercises and test.

Finally, the expert informed the objectives of the course and placed the order of the contents. Thus, having the main structures defined in the Course Definer Component, the mechanism was able to generate the XML Learning Scheme. Once the XML Learning Scheme was ready, we uploaded it to the Semantic Collaborative Environment proposed by Menolli, Reinehr and Malucelli [7]. A positive result presented is that the Semantic Collaborative Environment was able to read the generated XML Learning Scheme and look for learning objects related to the themes described on the Course Definer.

VI. FINAL CONSIDERATIONS

The work presented focuses on the identification of the specific needs inside an organization. The identified themes and roles that are presented to the expert are the basis for the definition of learning schema.

However, even a trained expert, who has sufficient knowledge to generate units of learning to assist the employees, has a complex job when it comes to know that the need for discovery by company employees, not only for growth of this

employee, but that this knowledge sustain it on a daily basis in the activities assigned to them.

Towards this direction of the problem, we propose a mechanism supported by concepts of organizational learning in order to contribute to the expert responsible for generating units of learning. This semi-automatic mechanism creates the units from the searches conducted on the Internet by means of search engines.

We estimated two main contributions to the completion of the proposed work. The first, is propose an approach to get the searches performed on search engines, aiming to catalogue these themes that were popular and along with a guiding company's knowledge, suggest topics of courses to be generated to the organization.

The second is to provide a component integrated into the semantic collaborative environment presented by Menolli, Reinehr and Malucelli [7], which is effective to help the expert generates the Units of Learning, in order to make the learning more effective.

Lastly, we identified some gaps that may be supplied on next works. The main gap is to run an experiment on an organizational environment, so other factors that surround the environment could be analyzed and evaluated. It would help identifying unanticipated problems and to adapt the mechanism to solve them. It is also identified the necessity of run another experiment with the objective of evaluate the Course Definer approach from a pedagogical perspective once the definition and application of a course is also related to the educational area.

REFERENCES

- [1] Neves, E. O., 2011. Organizational Learning: Considerations about methodologies of development promotions. *Administration and Economy University Magazine*, 3, 2 – 16.
- [2] Alvesson, M., 2000. Social identity and the problem of loyalty in knowledge-intensive companies. *Journal of Management Studies*, 37, n. 8, 1101-1123.
- [3] Menolli, A., Malucelli, A., Reinehr, S., 2011. Towards a Semantic Social Collaborative Environment for Organizational Learning in: *International Conference on Information Technology and Applications*, 65-70.
- [4] Nevis, E. C., Di Bella, A., Gould, J. M., 1995. Understanding organizations as learning systems. *Sloan Management Review*, 36, n. 2, 73-85.
- [5] Polsani, P. R., 2004. Use and abuse of reusable learning objects. *Journal of Digital Information*, 3, n. 4.
- [6] Koper, R., O., B., St.B.D., Ab.B. (2004). Representing the Learning Design of Units of Learning. *Educational Technology & Society*, 7, 97–111.
- [7] Menolli, A. L., Reinehr, S., Malucelli, A., 2013. Improving Organizational Learning: Defining Units of Learning from Social Tools. *Informatics in Education*. 12, n. 2, 273-290.
- [8] IMS Global Learning Consortium, 2003. "IMS Learning Design Information Model", Final Specification, from http://www.imsglobal.org/learningdesign/ldv1p0/imslid_infov1p0.html.
- [9] Amorim R. R., Lama M., Sánchez E., Riera A., Vila X. A., 2006. "A Learning Design Ontology based on the IMS Specification: The Need for a Learning Design Ontology," *Educational Technology & Society*, 38-57.
- [10] Rawlings, A., Van Rosmalen P., Koper R., Rodríguez-Artacho M., Lefrere P., 2002. "Survey of Educational Modelling Languages," *Learning Technologies Workshop*, from <http://www.cenorm.be/cenorm/businessdomains/businessdomains/iss/actvity/emlsurveyv1.pdf>.
- [11] Witten, I.H., Frank, E., 2005. *Data Mining: Practical Machine Learning Tools and Techniques*.
- [12] Ullman, J., Rajaraman, A., 2011. Mining of Massive Datasets, 2, 307-341.
- [13] Shishehchi, S., Banihashem, S. Y., Zin, N. A. M., 2010. A Proposed Semantic Recommendation System for E-learning, *ITSim*, 1, 1-5.
- [14] Gallego, D., Barra, E., G., A., H., G., 2013. Enhanced recommendation for e-learning authoring tools based on a proactive context-aware recommender, *Frontiers in Education Conference*, 1393-1395.
- [15] Leskovec, J., Rajaraman, A., Ullman, J.D., 2014. Mining of Massive Datasets, 2, 287-319.
- [16] Liang, G., Weining, K., Junzhou, L., 2006. Courseware Recommendation in E-learning System, *ICWL 2006, LNCS 4181*, 10-24.
- [17] Ricci F, Rokach L, Shapira B, Kantor PB., 2011. *Recommender Systems Handbook*. Springer.
- [18] Davenport, T. H., Prusak, L., 1998. *Working Knowledge: How Organizations Manage What They Know*. Boston, MA, USA: Harvard Business School Press.
- [19] Menolli, A. L., Cunha, M. A., Reinehr, S., Malucelli, A., 2015. "Old" theories, "New" Technologies: Understanding knowledge sharing and learning in Brazilian software development companies, *Information and Software Technology*, 58, 289-303.
- [20] Stan, J., Muhlenbach, F., LARGERON, C., 2014. Recommender Systems using Social Network Analysis: Challenges and Future Trends, *Encyclopedia of Social Network Analysis and Mining*, 1-22.
- [21] Reichling, T., V Veith, M., Wulf, V., 2007. Expert Recommender: Designing for a Network Organization, *Computer Supported Cooperative Work*, 16, 431-465.
- [22] Luo, Y., Cao, F., 2009. Web Knowledge Management System Based on User Model, *ICIE '09*, 1, 552-556.
- [23] Ale, M. A., Toledo, C. M., Chiotti, O., Galli, M. R., 2014. A conceptual model and technological support for organizational knowledge management, *Special Issue on Systems Development by Means of Semantic Technologies*, 95, 73-92.
- [24] Manning, C. D., Raghavan, P., Schütze, H., 2008. *An Introduction to Information Retrieval*, Cambridge University Press, 421-442.

Using peak analysis for identifying lagged effects between software metrics

Josée Tassé

Dept. of Computer Science and Applied Statistics
University of New Brunswick, Saint John campus
Saint John, New Brunswick, Canada
jtasse@unbsj.ca

Abstract— Measures extracted from software repositories tend to be collected on a regular basis (often daily), forming time series of data. In this context, it is normal to assume that some of the measures collected are having an effect on other measures, potentially with some delay (or “lag”) in the effect. Such delay in the effect may even vary over time, making the identification of the effect difficult. In this paper, we present our initial ideas on a simple analysis method for such situation, in which peaks from the two time series in question are analyzed, and similar ones are matched.

Keywords— Peak analysis; time series; lagged effect; software metrics

I. INTRODUCTION

Measures extracted from software repositories tend to be collected on a regular basis. For example, one could capture the number of bugs found and the number of bugs fixed every day or every week. In statistical terms, such sequence of data collected regularly over time form a time series.

Data are usually kept in this format (time series) only if they are simply displayed graphically (showing their evolution over time) or used in statistical control. For other analyses, data are typically transformed, losing most of their time-related information. In particular, one might aggregate such measures into a single one per version or per time segment (e.g., total number of bugs found, or total number of lines of code changed, during the development of one version). Then, only the differences between two consecutive versions or time segments are used in the analysis. For example, the expected number of bugs to be found during the development of a particular version may be predicted based on the number of lines of code changed in the previous version (and/or other variables of interest – those are surveyed in [1]). The problem here is that by limiting the association to only two consecutive time segments, it is not possible to detect a longer effect (e.g., what if a change in the number of lines of code changed was having an effect up to three releases later?).

What is needed is a way to analyze the metrics in question as time series, showing if one metric is causing another one to change, and if yes, after how much time can such impact be felt. Let’s illustrate this with the problem of judging the

stability of a system or sub-system. What is really needed here is to characterize what happened in the past (e.g. amount of change) that eventually lead to a decrease in the stability (perhaps measured as bugginess). Being able to analyze the lagged effect of the amount of change, or other metrics of interest, on the bugginess (or other indicators of stability), could help identify appropriate criteria for decisions related to software stability. If the lag is important enough to allow for a reaction time, this could lead to early warnings about upcoming problems with the stability unless something is done to reduce – or even eliminate – the problem. Depending on the type of changes done (e.g., corrective vs. perfective maintenance) or the complexity of the changes being made, it may take more or less time to feel such impact though. So the identification of the lagged effect is not a trivial issue.

Current statistical techniques, namely Granger causality [2] and transfer functions [3], mostly rely on building a regression model where the independent variables represent the same metric, but collected at some time in the past. For example, assuming that we would like to know the effect of time series x on time series y , with a maximum possible delay of 2 time periods, the following simple regression model could be built:

$$y(t) = a*x(t)+b*x(t-1)+c*x(t-2) \quad (1)$$

where $y(t)$ is the value of y at time t ; $x(t)$, $x(t-1)$, and $x(t-2)$ are the values of x at time t (no delay), at time $t-1$ (delay of 1), and at time $t-2$ (delay of 2) respectively; and a , b , and c are the regression coefficients. Note that more complex models can be built, including those where past values of y are used as well (for an autoregressive part).

The limitation with such a model is that it assumes that the delay in the effect is always constant, which is not always the case. Also, for a more general delayed effect such as “between 1 and 3 time periods”, one has to guess it and add the corresponding variable in the model being built. Trying all possible combinations of such kind of variable can be computationally expensive.

In this paper, we describe our initial work on a technique for solving such kind of problem. It relies on the identification of peaks in the two time series. Similar peaks across time series are then matched, showing at the same time the delay in the effect. The next section describes this technique with an

example, also comparing it with known statistical techniques. Section III provides information to help choose the right parameters for this technique. The last section provides other examples where we used this technique, as well as a discussion of the future work.

II. PEAK ANALYSIS

In order to build an approach for identifying lagged effect, an example for which the actual effect could be found in some other way was needed, to confirm its correctness. The following metrics were thus chosen for an initial analysis: number of bugs opened vs. number of bugs closed at each week during software evolution. There should be a lagged effect between them, as many of the open bugs eventually get closed.

We collected such data on JEdit, a Source-Forge project, over a period of 119 weeks. There were 229 bugs found during that period. We also extracted information on when each bug was opened and closed, and the duration between these two events. The distribution of these durations is clearly exponential: 50% of the bugs were fixed within the same week, 10% of them were fixed a week later, 10% of them fixed 2 to 4 weeks later, 10% between 5 and 10 weeks later, and the rest taking more than 10 weeks to be fixed, up to a maximum of 85 weeks.

We first checked our claim that other statistical techniques were not producing valuable results in our situation, by applying them to our data. The best model that could be built was the following (p-value = 0.000):

$$\text{close}(t) = 0.173 + 0.596 \text{ open}(t) + 0.331 \text{ open}(t-10). \quad (2)$$

Although the p-value was small, we had indications that the model was not good for prediction purposes: the R^2 value was only 39% (i.e., only 39% of the variation could be explained by the model), and the residuals were not normally distributed. This is not surprising, considering that only 50% of the bugs have a duration of 0, and less than 1% have a duration of 10. The reason for the difficulty in building a model is due to the fact that development happens in burst, and that the average duration (or lag) is changing over time. Our proposed approach is meant to overcome this issue.

The main idea behind our peak analysis technique is that if there is a lagged effect between two time series, peaks in one time series should match peaks in the other time series, with the lag being the time distance between the corresponding peaks. A peak here is defined as a time interval for which the value of the metric is significantly above its average value.

The first step in our proposed technique is to identify independently the peaks in the two time series. One key point in the definition of peak is the fact that we have to compare data with some average. However, one cannot assume that such average will stay the same over a long period of time. For example here, since development typically occurs in bursts, peaks during those bursts are expected to be higher than the peaks occurring during a

period of low activity. Still, we are interested in all peaks, not only the ones during bursts. Some kind of local average is thus needed. In statistics, this is handled through the concept of “moving average” [3]: a new time series is constructed using values corresponding to the average of a given number of consecutive values from the original time series. For example, assuming that we have a time series x , a new time series z corresponding to a moving average of 5 (later referred to as “MA5”) is composed of the following values:

$$z(t) = (x(t-2)+x(t-1)+x(t)+x(t+1)+x(t+2))/5 \quad (3)$$

for all $t=3$ to $n-2$ (n being the length of the time series x). For the values that cannot be calculated (i.e., at times $t = 1, 2, n-1$, and n), one can pad with the closest value that can be calculated (i.e., set $z(1)$ and $z(2)$ to the value of $z(3)$, and set $z(n-1)$ and $z(n)$ to the value of $z(n-2)$).

For peak identification, we actually build two new time series through moving averages: one to smooth the data, to be able to see a trend, and one to represent the local average. For our example here, we used an MA5 to smooth the data, and an MA49 (i.e., moving average where datapoints from $t-24$ to $t+24$ are averaged) for the local average. Padding is used to ensure that we have the same number of datapoints for the MA5 and the MA49. Figure 1 shows a plot of these moving averages for the count of opened bugs every week, and Figure 2 shows the one for the counts of closed bugs. Note that for space reasons, we show only an excerpt for weeks 30 to 90. Peaks are clearly appearing, as the intervals when the MA5 line is clearly above the MA49 line. In a more formal way, an interval is a peak only if it goes significantly above the MA49 line in at least one of its points. We use the standard deviation of the MA49 dataset as the threshold for being considered “significantly above the MA49”.

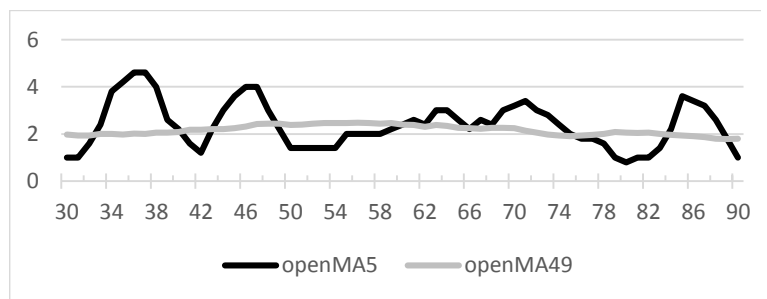


Figure 1. Plot of the MA5 and MA49 on opened bugs, showing peaks.

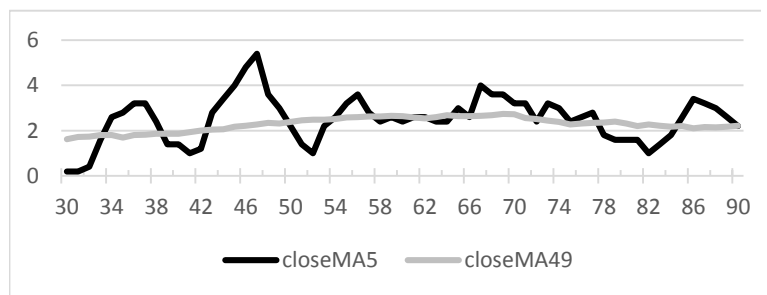


Figure 2. Plot of the MA5 and MA49 on closed bugs, showing peaks.

TABLE I. LOCATION AND SIZE OF PEAKS

Opened bugs			Closed bugs		
ID	interval	size	ID	interval	size
O1	33--40	12.2	C1	34--38	5.22
O2	43--48	5.98	C2	43--49	11.56
O3	60--75	7.7	C3	54--57	1.88
O4	84--89	5.56	C4	65--77	6.68
			C5	85--89	4.02

Table I lists those peaks, with their time interval and their size. The size is calculated as the sum of the differences between the MA5 and the MA49, for all points in the interval. Note that in the case where two peaks are separated by only one data point falling slightly below the MA49 (by less than the standard deviation on the MA49 dataset), those two peaks are merged into one. The third peak for open bugs is such a merged peak, covering from time 60 to 75, in spite of a dip at time 66.

The second step of our approach is to match the identified peaks in the two time series. The idea is to match peaks of similar size sequentially, trying to match as many peaks as possible. The matches have to be valid though: the start time of the close peak should be no earlier than the start time of the matched open peak, and same for the end times. In our example, the best set of matches is as follows: C1 unmatched, O1 matched to C2, O2 matched with C3, O3 matched to C4, and O4 matched to C5.

We evaluate how good the set of matches is by calculating the Pearson correlation between the corresponding sizes. When a peak is unmatched (e.g., C1 here), we associate it with a peak of size zero. In the example above, the correlation between the vectors [0.0, 12.2, 5.98, 7.7, 5.56] and [5.22, 11.56, 1.88, 6.68, 4.02] is 0.65. This is the best possible correlation for the data above. The algorithm for finding such best set of matches builds all possible sets using a backtracking approach, where the following conditions apply: for all peak O_i matched to a peak C_j , (a) each open peak earlier than O_i is either not matched or matched to a close peak earlier than C_j , and (b) each open peak later than O_i is either not matched or matched to a close peak later than C_j . For each possible set of matches, the correlations is calculated, and the set corresponding to the best correlation is kept.

In the best set of matches identified above, one can see that many of the matched peaks are very close in size (i.e., O1—C2, O3—C4, and O4—C5). However, the peak C1 is quite large for an unmatched peak, and the matched peaks O2—C3 are very different in size. This is why the correlation is quite weak. Looking at the graphs again (Figures 1 and 2) and the sizes of the peaks in Table I, one can see that the peak C1 could

TABLE II. MATCHED PEAKS AND HALF-PEAKS

Opened bugs			Closed bugs		
ID	interval	size	ID	interval	size
			C1	34--38	5.22
O1	33--40	12.2	C2	43--49	11.56
O2	43--48	5.98	C3	54--57	1.88
O3	60--75	7.7	C4	65--77	6.68
O4	84--89	5.56	C5	85--89	4.02

actually be somewhat covered by the peak O1, with the remaining of O1 covered by C2 (C2 is large enough to cover both O2 and part of O1). In order to spot these possibilities, the algorithm above can be repeated using half peaks rather than full peaks, in trying to identify the best matches: each peak is cut in half, and considered as its own separate peak, cutting in half the original interval and size. The correlation is still done using the full open peaks though, adding the corresponding half peaks when possible. Table II shows the results for the half-peak analysis. Corresponding peaks are the ones on the same row in the table. The correlation in this case is really high at 0.969 (between the vectors [0.0, 12.2, 5.98, 0.0, 7.7, 5.56] and [5.22/2, 5.22/2+11.56/2, 11.56/2, 1.88, 6.68, 4.02]).

We also tried to see if we could detect that one variable does not have an effect on the other. In the case here, we would not expect the count of close bugs to lead to a later effect on the count of open bugs, as very few of the close bugs get reopened. So we ran our algorithm again, reversing the two time series. In this case, the best correlation was only 0.27 (for the whole 119 weeks of the example, not just the excerpt presented here). The half-peak analysis could improve this to 0.75, but this is still too low to be considered a true effect.

The obvious question now is: does this correspond to what is actually going on with the bugs in those time intervals? The answer is yes. For all of the peaks O2 to O4, between 70% and 82% of the bugs were closed within the interval of their matched peak. In the case of the peak O1, 43% of its bugs were associated to C1, while 23% of them were associated with C2.

We also looked at the average durations of the bugs (i.e., number of weeks between the time they are opened and the time they are closed) within each of the open peaks. These were 9.07, 10.65, 3.88, and 2.41 for peaks O1 to O4 respectively. This seems in line with our initial findings: peak O1 has a much higher average duration due to its match to a peak that is at a distance of 9 to 10 weeks. It still contains a large percentage of bugs fixed almost immediately (i.e., duration of 0 or 1). The peak O3 is matched to a peak that is at a distance of 2 to 5 weeks, matching its average duration. The average duration for the peak O4 is slightly larger than its distance to its matched peak, but this peak does contain a much larger percentage of bugs fixed immediately than the entire set of bugs (71% vs. 51%). For the peak O2, its average duration does not seem to match its distance to its matched peak. This is due to the fact that it contains a much larger percentage of bugs that are fixed within a very long time frame, with a large spread in duration. Actually, we could have seen the problem in the first place in our analysis, if we had considered the second best set of matches: if we match O2 to both C2 and C3, the correlation is still really high at 0.953. And, the distance between peaks O2 and C3 is approximately 10, which does correspond to the average duration.

As one can see, such an analysis can show how the distribution of bug durations changed over time. Using such information, one could try to identify what exactly happened at those times to cause such differences (e.g., perhaps the kind of maintenance – corrective vs. perfective – was different). We did not have enough knowledge of this particular software evolution (of JEdit) to perform such investigation.

III. CHOICE OF VALUES FOR MOVING AVERAGES

In the technique as presented above, two moving averages are used (one to smooth the data and one to act as a local average) for identifying peaks. In the example provided, moving averages of 5 and 49 were used. Other numbers can be used as well, depending on what exactly one would like to find out.

The analysis of changes in the lagged effect over time (as mostly performed above) does not help identify the general lag in the overall effect. Increasing the size of the moving average used can help with this. For example, using the context and data above but a moving average of 9 rather than 5, we get 3 peaks that are matched to 3 other peaks of roughly the same size. The apparent lag in this case is between 0 and 3, which corresponds to the duration of 70% of the bugs. By doing this though, we lose the possibility of identifying the times when such lag was different. On the other hand, if we want to be even more specific in the lags and the differences at particular times, a smaller moving average (e.g., MA3) can be used.

By increasing the size of the moving average, the length of the intervals for the peaks found increases, and the number of such peaks is reduced. Then, when a match is found, the peaks tend to be of more similar sizes. However, because of the longer interval, there could be more bugs that have a much higher duration than what would be seen when looking at the distance between the matched peaks. For example, if matched peaks are located in the interval [30..45] and [31..46], although the apparent delay is just 1, there could still be many bugs with a much higher duration (e.g., a bug being opened at time 30 but closed at time 45).

For the size of the moving average used as the local average (e.g., the MA49 used in the previous section), our experience shows that an appropriate number should be 5 to 10 times larger than the moving average used to smooth the data.

IV. VALIDATION AND DISCUSSIONS

In Section II above, we have shown how our technique works on a subset of the data we had from the open source software JEdit. It should be noted that the technique was successful for the entire 119 weeks of data that we had, which included an extra matched peak prior to week 30, and an extra matched peak after week 90 (not displayed above).

We validated our technique on a second open source system: MinGW (another SourceForge project). We extracted the same kind of data as described in Section II above, over a period of 175 weeks. There were 203 bugs found during that period. Through our technique (using an MA5 vs. an MA49 as described above), we could identify 11 peaks for opened bugs and 11 peaks for closed bugs. Those peaks were very diverse, with interval lengths ranging from 2 to 14 weeks, and with sizes ranging from 0.2 to 9.5. The duration of the bugs was somewhat longer than the ones found in JEdit, with the following distribution: 27% of the bugs were fixed within the same week, 26% of them were fixed a week later, 13% of them fixed 2 to 4 weeks later, 14% between 5 and 10 weeks later, and the rest taking more than 10 weeks to be fixed, up to a maximum of 123 weeks. We successfully matched the related

peaks with a correlation of 0.84 (0.90 when improving it using half-peaks). In almost all cases, the majority of the bugs (62% to 100%) opened within a given peak were closed within the matching peak. There was one exception, but this was with a relatively small peak, matched to a much larger peak (i.e., with very different sizes).

From that system (MinGW), we also analyzed the time lag between the number of commits per week and the number of opened bugs per week (i.e., how long does it take in general to find bugs after modifications are made). We saw that such lag was approximately 8 to 9 weeks, with one exception where the lag was only 3 to 5 weeks. That time corresponded with an increase in new features developed. Such kind of information can be useful in planning when there will be an increase in demand for fixing bugs.

We also used our technique to confirm previous results in another (unrelated) project, where we have shown that making many highly-dispersed changes in a file was increasing the risk of finding a bug in that file within three months after the change [4]. This previous work was looking at individual files going through a sudden burst of changes, characterizing those changes and predicting the bugginess of the file based on similarities with past cases. Here, we looked at the proportion of the file commits performed every week, for all files rather than individual files, that were implementing highly-dispersed changes. We compared this with the number of bugs detected every week. And indeed, the peaks in these two time series were matching, with a typical lag between 2 and 7 weeks. This confirmed our previous results, with even more precise information about the lag. We repeated the work on other types of changes (e.g., small local change, massive local change), but we could not see a match in the peaks. This supported our previous findings too, that other types of changes were either not affecting the bugginess of the file, or were affecting it only when the file was large. Not only did we confirm previous results here, but such analysis could help building a better bug predictor.

As future work, further validation is clearly required. At this point, we tried our approach on bug data for only two systems: JEdit and MinGW. We need to try it out on more (various) systems, over a longer period of time, and on other kinds of metrics that could be analyzed this way. Improvement to the underlying algorithm is also necessary in order to make it more efficient, and practical for larger inputs. Finally, we are interested in applying this kind of technique to areas other than software engineering.

REFERENCES

- [1] M. D'Ambros, M. Lanza, R. Robbes, "An Extensive Comparison of Bug Prediction Approaches", Proc. of the 7th IEEE Working Conf. on Mining Software Repositories, Cape Town, South Africa, May 2010, pp. 31-41.
- [2] C.W.J. Granger, "Testing for causality: A personal viewpoint", Journal of Economic Dynamics and Control, vol. 2, pp. 329-352, 1980.
- [3] D.C. Montgomery, C.L. Jennings, and M. Kulahci, Introduction to Time Series Analysis and Forecasting, Wiley, 2007.
- [4] J. Tassé, "Using code change types in an analogy-based classifier for short-term defect prediction", 9th Int. Conf. on Predictive Models in Software Engineering, Article No. 5, Baltimore, Maryland, Oct. 2013.

Integration of Software Measurement Supporting Tools: A Mapping Study

Vinícius Soares Fonseca

Monalessa Perini Barcellos

Ricardo de Almeida Falbo

Ontology and Conceptual Modeling Research Group (NEMO)
Computer Science Department, Federal University of Espírito Santo
Vitória, ES, Brazil
{vsfonseca, monalessa, falbo}@inf.ufes.br

Abstract - During software projects, it is necessary to collect, store and analyze data to support decision making at project and organizational levels. Software measurement is a key practice to process quality improvement and project management. Given the nature of measurement activities, supporting tools are essential. Different tools can be combined to support the measurement process and provide the necessary information for decision making. However, usually these tools are developed by different developers, at different points in time and without concern for integration. As a result, organizations have to deal with integration issues to allow tools communication and properly support the measurement process. This paper presents a study investigating in the literature initiatives involving tools integration to support software measurement. As a result, twelve proposals were analyzed and their characteristics are presented.

Keywords - Software measurement; software measurement process; software measurement tools; integration; interoperability; systematic mapping

I. INTRODUCTION

Organizations use software measurement in several contexts. For example, in project management, measurement helps to develop realistic plans, monitor progress, identify problems and justify decisions [1]. In process improvement, measurement supports analyzing process behavior, identifying needs for improvement and predicting if processes will be able to achieve the established goals [2]. For this, data related to software processes, such as project management and testing processes, must be collected and analyzed.

Typically, organizations use different tools to support different processes, such as supporting tools for project management, requirements management, testing and bug tracking. Although, in general, these tools are not conceived aiming at supporting software measurement, many times they support collecting and storing useful data related to those processes (e.g., number of detected defects, time spent on activities, number of lines of code, etc.).

In order to properly support the software measurement process, providing consistent data to generate useful information, tools should be integrated. However, integration is a complex task. In general, each tool runs independently and implements its own data and behavioral models, which are not shared between different tools, leading to several conflicts [3].

(DOI reference number: 10.18293/SEKE2015-058)

Considering this scenario, we investigated the literature looking for initiatives involving tools integration to support software measurement. Aiming to reduce bias and ensure the study repeatability, the investigation was conducted as a systematic mapping. According to [4], a systematic mapping makes a broad study in a topic of a specific theme and aims to identify available evidence about that topic.

This paper is organized as follows: Section II briefly discusses software measurement and integration; Section III presents the research protocol used to guide the study; Section IV describes the obtained results; Section V discusses the results; and Section VI presents our final considerations.

II. BACKGROUND

A. Software Measurement

Software measurement (SM) is a primary support process for managing projects. It is also a key discipline in evaluating the quality of software products and the performance and capability of software processes. The software measurement process includes: measurement planning, measurement execution, and measurement evaluation [5].

For performing software measurement, initially, an organization must plan it. Based on its goals, the organization has to define which entities (processes, products and so on) are to be considered for software measurement and which of their properties (size, cost, time, etc.) are to be measured. The organization has also to define which measures are to be used to quantify those properties. For each measure, an operational definition should be specified, indicating, among others, how the measure must be collected and analyzed. Once planned, measurement can start. Measurement execution involves collecting data for the defined measures, storing and analyzing them. The data analysis provides information to decision making, supporting the identification of appropriate actions. Finally, the measurement process and its products should be evaluated in order to identify potential improvements [6].

B. Integration and Interoperability

Integration and interoperability are very related notions. *Integration* can be defined as the act of incorporating components into a complete set, conferring it some expected properties and creating synergy [3]. *Interoperability*, in turn, can be understood as the ability of applications or application components to exchange data and services [7]. Due to their

interrelation, these terms are often used in an indistinct way [8]. In this paper, the term *integration* is adopted with a wider sense, covering both integration and interoperability meaning.

Izza [3] synthesizes integration approaches through four main dimensions: *scope*, which distinguishes between intra- and inter-enterprise integration; *viewpoint*, considering user, designer, and programmer views; *layer*, referring to data, service/message, and process integration; and *level*, which considers hardware, platform, syntactical, and semantic integration. For this paper, the two last dimensions are particularly relevant. Regarding integration layers, data integration deals with moving data between multiple data stores. Integration at this layer assumes bypassing the application logic and manipulating data directly in the database, through its native interface. Service/message integration addresses messages exchange between the integrated applications. Process integration views enterprises as a set of interrelated processes and it is responsible for handling message flows, implementing rules and defining the overall process execution. With respect to integration levels, syntactical integration encompasses the way data model and operation signatures are written down, while semantic integration encompasses the intended meaning of the concepts in a data schema or operation signature [3].

Challenges in applications integration arise, among others, from the fact that heterogeneous applications employ different data and behavioral models, leading to semantic conflicts. These conflicts occur whenever applications are built with different conceptualizations, which can impact the integration of data, services, and processes [8].

III. THE RESEARCH PROTOCOL

The study was performed following the approach defined in [4]. According to this approach, a systematic mapping involves: *planning*, when the research protocol is defined; *conducting*, when the protocol is executed and data

are extracted, analyzed and recorded; and *reporting*, when the results are recorded and made available to potential interested parties. In this section we present the main parts of the research protocol used to perform the study.

Research Questions: the goal of this mapping study is to depict a general view of the current status of the research regarding tools integration to support software measurement. Table I presents the research questions that this mapping study aims to answer, as well as the rationale for considering them.

Search String: the search string was developed considering three groups of terms that were joined with the operator AND. The first group includes terms related to integration and interoperability. The second group includes terms related to software measurement. The third group includes terms related to tools and applications. Within the groups, we used the OR operator to allow synonyms. The following search string was used: ("integration" OR "integrated" OR "interoperability" OR "interoperable") AND ("software measurement" OR "software process measurement" OR "software project measurement" OR "software engineering measurement" OR "software product measurement") AND ("tool" OR "application" OR "system" OR "framework" OR "suite" OR "toolkit").

For establishing this search string, we performed some tests using different terms, logical connectors, and combinations among them. More restrictive strings excluded some important publications identified during the informal literature review that preceded the systematic mapping. These publications were used as control publications, meaning that the search string should be able to retrieve them. We decided to use a comprehensive string that provided better results in terms of number and relevance of the selected publications, even though it had selected many publications that had to be eliminated in subsequent steps.

TABLE I. RESEARCH QUESTIONS

ID	Question	Rationale
RQ1	When and in which type of vehicle (journal / scientific event) have the publications been published?	Give an understanding on when and where the selected publications have been published.
RQ2	Which types of research have been done?	Identify the research type according to the classification defined by Wieringa et al. [9]: Evaluation Research; Proposal of Solution; Validation Research; Philosophical Paper; Opinion Paper; and Personal Experience Paper.
RQ3	Which types of tools have been integrated for supporting SM?	Identify the types of the integrated tools (e.g., project management tool, issue tracking tool, etc.) and verify whether a type is used in more than one proposal.
RQ4	Have the integrated tools been developed by the same group or organization?	Verify whether or not the initiatives have been integrating tools developed by the same group or organization. The purpose is to analyze if there is a trend in using tools developed by the same or by different groups.
RQ5	Which SM process activities (measurement planning, data collection, and data analysis) are supported by the integrated set of tools?	Identify which measurement activities are being supported by the initiatives, in order to evaluate the coverage of the resulting set of integrated tools. The activities considered are the two first activities established in [5] (measurement planning and measurement execution). Moreover, measurement execution was split for allowing us to verify if the tools support both data collection (which involves data collection itself and data storage) and data analysis, or only one of them.
RQ6	Which categories of measures are addressed by the proposal?	Identify which categories of measures (e.g., code measures, tests measures, etc.) have been considered, allowing us to analyze how specific or comprehensive is the measurement scope.
RQ7	In which layers (data, message/service or process) does the integration occur?	Identify the layers in which the integration is performed, considering the layers defined in [3]. The purpose is to analyze in which layer the integration initiatives have been focused on.
RQ8	In which level (syntactical or semantic) does the integration occur?	Identify the levels in which the integration is performed, considering the levels defined in [3]. The purpose is to analyze in which level the integration initiatives have been focused on.
RQ9	Does the proposal support measurement in the context of maturity models or standards? If so, which ones?	Identify which proposals support measurement in the context of maturity models (e.g., CMMI) or standards (e.g., ISO/IEC 9001), allowing us to verify whether supporting maturity models and standards has been a concern in SM tools integration initiatives.

Sources: the following six electronic databases were searched: IEEE Xplore (<http://ieeexplore.ieee.org>), ACM Digital Library (<http://dl.acm.org>), Springer Link (<http://www.springerlink.com>), Scopus (<http://www.scopus.com>), Science Direct (<http://www.sciencedirect.com>), and Engineering Village (<http://www.engineeringvillage.com>). They were selected based on other systematic reviews in the Software Engineering area.

Publication Selection: selection was performed in five steps:

Step 1 (S1) - Preliminary selection and cataloging: the search string was applied in the search mechanisms of the selected sources. Publication type was limited to papers from the Computer Science and Engineering area.

Step 2 (S2) – Duplicates Removal: studies indexed by more than one digital library were identified and the duplications were removed.

Step 3 (S3) – Selection of Relevant Publications – First Filter: selecting publications by applying a search string does not ensure that all selected publications are relevant, because such selection is restricted to syntactic aspects. Thus, the title, abstract and keywords of the selected publications were analyzed considering the following inclusion (IC) and exclusion (EC) selection criteria: (IC1) the publication presents information regarding integration among tools, applications or systems that support software measurement; (EC1) the publication does not have an abstract; (EC2) the publication is published as an abstract; and (EC3) the publication is not a primary study.

Step 4 (S4) - Selection of Relevant Publications – Second Filter: the full text of the publications selected in S3 was read with the purpose of identifying the ones that provide useful information. Thereby, the inclusion criterion IC1 was considered and also the following exclusion criteria: (EC4) the publication is not written in English; (EC5) the publication full text is not available; and (EC6) the publication is a copy or an older version of an already considered publication.

Step 5 (S5) - Snowballing: as suggested in [4], the references of publications selected in the study must be analyzed and, if some of them seems to present evidence related to the research topic, it should be assessed by the selection criteria and included in the study. Thus, in this step, references of the publications selected in S4 were investigated by applying the first and second filters.

IV. RESULTS

The systematic mapping considers publications until December 31st 2014. As a result of S1, 948 publications were obtained (357 from IEEE Xplore, 90 from Scopus, 257 from ACM, 8 from Science Direct, 49 from Engineering Village and 187 from Springer Link). After S2, 85 duplications were

eliminated, achieving 863 publications. After S3, only 24 studies were selected (a reduction of approximately 97%). After S4, we achieved 8 studies. Applying snowballing (S5), 4 publications were added, reaching a total of 12 publications. Table II presents a brief description of each proposal. Following, we present the main results obtained for each research question.

Publications source and year (RQ1): as Table II shows, the first study was published in 1988. Some studies were published since then, but research in the area has not been stable and presents two gaps (one between 1989 and 1996, and another between 2004 and 2009). Since 2010, it seems to be a more continuous research in the topic, with at least one work published per year. Most studies were published in scientific events (7) instead of journals (5).

Research type (RQ2): all the analyzed studies include *proposals of solution*. Studies [11], [12], [13], [14], [15], [19], [20] and [21] are also categorized as *evaluation research*, since they have been applied into a production environment in at least one organization. Studies [10], [16] and [18] are also considered *validation research* due to the use of a prototype or experiment to evaluate the proposal.

Integrated Tools (RQ3, RQ4): Table III presents the types of the tools being integrated in each proposal. Except [10], in which it was not possible to identify whether the tools were developed by the same group or not, all proposals integrated tools developed by different groups.

Measurement activities and Measures (RQ5, RQ6): only four proposals ([10], [12], [13] and [20]) address *Measurement Planning*. *Data Collection* and *Data Analysis*, in turn, are addressed by all proposals. Table III presents the categories of the measures addressed by the proposals.

Integration layers (RQ7): seven proposals ([12], [13], [14], [15], [18], [20] and [21]) address integration only in *data layer*; four proposals ([10], [16], [17], and [19]) address integration only in the *message layer*; and one proposal ([11]) addresses integration in both *data* and *message layers*. None of the proposals address the *process layer*.

Semantic aspects (RQ8): only [16] addresses integration in the semantic level; the others address integration in the syntactical level. In [16], Ghezzi and Gall use ontologies implemented in OWL to define and represent the data consumed and produced by the services of the integrated tools.

Maturity Models/Standards (RQ9): only two studies mention maturity models/standards and both of them refer to CMMI [22]. [15] uses CMMI measurement practices to define the measurement program supported by the proposed framework. [20] was conducted at a CMMI Level 3 organization.

TABLE II. PROPOSALS INVOLVING TOOLS INTEGRATION FOR SUPPORTING SOFTWARE MEASUREMENT

Proposal	Year/Vehicle	Description
TAME [10]	1988 Journal	TAME (Tailoring A Measurement Environment) system is an Integrated Software Engineering Environment that is composed by several components. TAME integrates three measurement tools that capture data from Ada source and generate measures.
Tool support for SM [11]	1997 Journal	This approach uses a set of integrated tools in order to support software measurement and quality improvement. A tool that supports tree-modeling analysis (S-PLUS) is the central analysis tool. Other tools are used for data gathering, analysis, and result presentation. The tools are connected to S-PLUS, either as an information consumer or as information providers. The integration is achieved through the adoption of external rules for data contents and formats (to ensure tools interoperability), the usage of common tools for multiple purposes, and the usage of utility programs that convert data for tools interoperability.
GQM tool [12]	2000 Journal	GQM tool supports measures definition, data collection, analysis, and feedback. It has interface with a configuration management system and other measurement tools. The integration with the configuration management system is performed through a data link between their relational databases. The integration with the other measurement tools is done by developing an XML translator for each tool, allowing the translation of the native data format to XML format.
MetriFlame [13]	2001 Symposium	MetriFlame is a measurement automation tool based on GQM that uses existing data recorded during software development process. It has components for collecting and converting measurement data from various tools, spreadsheets and databases. In the paper, the components are not detailed and no further information about integration is given.
A Decision Support System [14]	2003 Workshop	The Decision Support System was developed at IBM for tracking and using software measures aiming to enable executives to make better informed decisions in supporting their products. It captures (from different host systems) data regarding customer support, critical situations and customer satisfaction and integrates these data into a data warehouse.
SM in a CI Environment [15]	2010 Conference	It uses a Continuous Integration (CI) ¹ engine in order to automate measurement data extraction. It follows CMMI Measurement and Analysis process area practices and GQ(I)M concepts for selecting relevant measures. Data collection is done by several tools. After extraction, data are consolidated in a XML file that is stored into a relational database until an ETL (Extract, Transform, and Load) process run and load data into a data warehouse. An OLAP tool is used to analyze data.
SOFAS [16]	2011 Conference	SOFAS is a platform that offers software analysis services to allow for interoperability among analysis tools. It is made up of three main constituents: Software Analysis Web Services, which provides a catalogue of services for data analysis; a Software Analysis Broker, acting as the services manager and the interface between the services and the users; and Software Analysis Ontologies, which defines and represents the data consumed and produced by the different services.
Dione [17]	2012 Symposium	Dione is a Java web application whose majors functions are: i) build a measurement repository that contains product and process measures as well as information about defected software components; ii) analyze trends in measures and issues using chart and report configurations; and iii) construct and calibrate customized defect prediction models to predict defect proneness of a software product version or release. It collects data from several tools and uses a smart client to connect with software development artifacts and automatically extract measures. It also supports integration with other tools through web services.
QualitySpy [18]	2012 Journal	QualitySpy is a framework for monitoring the software development process. It collects raw data from several integrated tools as well as from the source code. The collected data and reports are available in a reporting module implemented as light web client, which communicates with the server using Representational State Transfer (REST).
The 3C Approach [19]	2012 Workshop	The 3C Approach is an extension to the CI practice and addresses Continuous Measurement and Continuous Improvement as subsequent activities to Continuous Integration. Several Java tools and a version control system were integrated into the CI engine CruiseControl, allowing collection of measures related to source code and test coverage.
ASSIST [20]	2013 Conference	ASSIST is an integrated tool developed by a CMMI level 3 organization. It adopts GQ(I)M approach and is connected with commercial software suites for project management, issue tracking and enterprise resource planning. ASSIST uses a low-level integration strategy based on SQL because all the tools involved depend on a relational database management system. It allows automatic data collection from the integrated tools, data import from spreadsheets and manual data entry.
DePress [21]	2014 Journal	DePress is an open source, extensible framework for software measurement and data integration which can be used for prediction purposes (e.g., defect prediction, effort prediction) and software changes analysis (e.g., release notes, bug statistics). It supports the integrated use (through KNIME Framework) of the issue tracking systems JIRA and Bugzilla, the software configuration management systems SVN and GIT, and the measurement tools Judy, JaCoCo, EclipseMetrics, CheckStyle and PMD.

TABLE III. TYPES OF INTEGRATED TOOLS AND CATEGORIES OF MEASURES ADDRESSED BY THE INTEGRATION INITIATIVES.

Pub.	Types of the tools being integrated	Measure categories
[10]	Code Measurement, Tests	code, size, test
[11]	Code Measurement, Tests, Configuration Management, Issue Tracking, Modeling, Presentation/Reporting, Reverse Engineering	code, size, test, defects, changes, design, transactions
[12]	Code Measurement, Configuration Management	code, size, defects
[13]	Configuration Management, Project Management, Document Management, Databases, Spreadsheets	it depends on the data available in the integrated tools, databases and spreadsheets
[14]	Customer Management, OLAP Tool	problem, product quality, customer satisfaction
[15]	Code Measurement, Tests, Continuous Integration, Build Automation, OLAP Tool	code, size, test
[16]	Code Measurement, Configuration Management, Issue Tracking	code, size
[17]	Code Measurement, Configuration Management, Issue Tracking, Presentation/Reporting	code, size, defects
[18]	Code Measurement, Configuration Management, Issue Tracking, Continuous Integration	code
[19]	Code Measurement, Tests, Configuration Management, Continuous Integration, Build Automation, Presentation/Reporting	code, size, test
[20]	Issue tracking, Project management, Enterprise Resource Planning, Spreadsheets	code, size and other measures depending on the data available in the integrated tools and spreadsheets
[21]	Code Measurement, Configuration Management, Issue Tracking, Presentation/Reporting, Defect Prediction, Data Mining Tool, Security, Statistics, Spreadsheets	code, defects, time, issue

¹CI is a practice for continuous integration of new source code into the base code, including automated compile, build and running of tests [19].

V. DISCUSSION

This section provides some additional discussion about the results presented in the previous section.

Concerning the measurement activities, we noticed that all proposals that support *Measurement Planning* ([10], [12], [13] and [20]) use GQM (Goal-Question-Metric) paradigm [23] or one of its variations. GQM is based on the idea that measurement should be goal-oriented, i.e., data must be collected based on an explicitly documented rationale [12]. Thus, GQM addresses measurement planning by guiding measures definition from established goals. Since GQM has been successfully adopted in software measurement initiatives for years, its usage by the proposals that address measurement planning was expected.

All proposals support *Data Collection* and *Data Analysis*. Regarding data collection, data are automatically captured by the tools or input by using their interface. As for data analysis, [11] supports the use of collected data to analyze software reliability; [10], [12], [13], [17] and [19] allow to analyze whether the established goals have been achieved; [14] supports the analysis of customer satisfaction; and [17] and [21] support analysis aiming at defect prediction.

With respect to the integrated tools, although we did not list in this paper the tools involved in each proposal, they are diverse. There are proposals integrating commercial tools (e.g. [11], [13], [20]), open source tools (e.g. [15], [16], [18], [19], [21]) and in-house developed tools (e.g. [10], [12], [20]). We also noticed that some proposals focus specifically on integrating existing tools (e.g. [11], [15], [16], [18], [19]), while others address the development of a whole integrated tool (e.g. [10], [20]). Moreover, we noticed that in some initiatives ([11], [12], [13], [15], [16], [20], [21]) measurement process support was the main motivation for integrating tools, while in others ([10], [14], [17], [18], [19]) the measurement support was achieved as a consequence of the tools integration. For instance, in [18] tools are integrated to support software development process monitoring and, as a consequence of the integration, software measurement was also supported. The variety of tools that can be used to support measurement increases the relevance of integration in this domain, because organizations could choose the tools that are more suitable for their needs and work on their integration.

Even though the integrated tools are diverse, it is possible to notice a predominance of code-related tools. Code Measurement, Issue Tracking, and Configuration Management tools are integrated in several proposals (9, 7 and 6 proposals, respectively). It might be a consequence of these types of tools being prone to automatic collection of measures. Besides, some of them depend on others to provide information. For instance, since source code is usually stored in a Configuration Management system, the presence of a Code Measurement tool usually implies the presence of a Configuration Management tool.

Considering that code-related tools were integrated in most proposals, it was expected that code measures (e.g., cyclomatic complexity, number of methods) would be addressed by most proposals. 10 of the 12 studies address them. Taking the types of integrated tools and measures into account, except [13] and [20], which have a more comprehensive scope, the integration initiatives usually address a specific measurement scope (e.g. coding, customer support).

Analyzing the integration layers addressed, we noticed that the proposals deal with *data* and *message layer*, while *process layer* is not addressed. We believe this is due the fact that *process layer* integration (commonly referred to as Business Process Integration) constitutes the most complex integration approach [3]. It views an enterprise/organization as a set of interrelated business processes and not merely islands of information, dealing with message flows, rules and process execution. *Message layer* is addressed, but only by few proposals. Message layer integration requires tool communication by means of message exchange between the tools. If the integrated tools are not able to communicate by means of messages, integration in this layer demands extra effort, especially if tools were not developed by the group performing the integration (this is the case in most proposals). In this sense, according to [20], a low level of integration is preferred when integrating with commercial tools, because it does not involve any code development or modification at the commercial tools' side. All studies that integrate commercial tools ([11], [12], [13], [20]) are limited to data integration.

As for semantic integration, only [16] considers semantic aspects in the integration. Neglecting semantics during an integration initiative is a serious issue, since many semantic problems can occur, such as the ones called "false agreement", which are described in [28] and include: the use of equivalent terms with different meaning; the use of equivalent terms with partially equivalent meaning; the use of different terms with equivalent meaning; and the use of different terms with a certain degree of equivalence. For addressing these problems, ontologies can be used to establish a common conceptualization about the domain in order to support communication and tools integration [24].

Since none of the proposals presented the method followed to perform the integration, we concluded that they have used ad-hoc approaches for integrating the tools. Not using a systematic approach for performing the integration, despite the existence of systematic approaches in the literature (e.g. [24], [25], [26]), can be seen as a gap regarding methodological aspects. Systematic approaches can structure the integration process into different levels of abstractions and define guidelines on how to perform the various integration activities. This is essential for establishing an engineering approach for application integration [27].

VI. FINAL CONSIDERATIONS

This paper presented the main results of a systematic mapping about initiatives involving tools integration for supporting software measurement. A total of 952 publications

were analyzed and 12 proposals involving tools integration for supporting software measurement were found.

According to [29], a mapping study gives an idea of shortcomings in existing evidence, which becomes a basis for future studies. Therefore, the main objective of this mapping was to analyze the proposals and provide a general view of the current status of the research regarding tools integration for supporting software measurement. Summarizing, the analyzed proposals address measurement execution (data collection and analysis), but most of them do not address measurement planning. Integration in the data layer is most common, although some proposals deal with integration in the message layer. They predominantly integrate coding-related tools and address code measures. Supporting maturity models/standards have not been a concern. Finally, only one proposal considers semantic aspects and, apparently, none of the proposals used a systematic approach to perform integration.

This panorama reveals some gaps in the research regarding tools integration for supporting software measurement. We can highlight the following: (i) lack of concern with semantics; (ii) limited coverage with respect to the measurement process or the measure categories addressed by the integrated tool suite; (iii) lack of alignment to quality-related standards and maturity models; (iv) failure to consider a holistic view of the (software) process, leading to the absence of integration in the process layer. Taking these gaps into account, we are now working on an integration of measurement supporting tools following a systematic approach aiming at overcoming these gaps.

ACKNOWLEDGMENT

This research is funded by the Brazilian Research Funding Agency CNPq (Processes 485368/2013-7 and 461777/2014-2).

REFERENCES

- [1] J. McGarry, D. Card, C. Jones, B. Layman, E. Clark, J. Dean, and F. Hall, "Practical Software Measurement: Objective information for decision makers," Addison Wesley, Boston, USA, 2002.
- [2] W. A. Florac, A. D. Carleton, "Measuring the software process: statistical process control for software process improvement," Addison Wesley, Boston, USA, 1999.
- [3] S. Izza, "Integration of industrial information systems: from syntactic to semantic integration approaches," *Enterp. Inf. Syst.*, vol. 3, no. 1, pp. 1–57, Feb. 2009.
- [4] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," TR EBSE-2007-01, School of Computer Science and Mathematics, Keele University, 2007.
- [5] ISO/IEC 15939, "Systems and Software Engineering – Measurement Process", 2007.
- [6] M. Barcellos, R. A. Falbo, and A. R. Rocha, "Establishing a well-founded conceptualization about software measurement in high maturity levels," in *Proc. of the 7th International Conference on the Quality of Information and Communications Technology*, 2010, pp. 467–472.
- [7] P. Wegner, "Interoperability," in *ACM Computing Survey*, 28 (1), 1996, pp. 285–287.
- [8] J. C. Nardi, R. A. Falbo, and J. P. A. Almeida, "A panorama of the semantic EAI initiatives and the adoption of ontologies by these initiatives," in *Proc. of the IWEI 2013, LNBP 144, Lecture Notes in Business Information Processing*, 2013, pp. 198–211.
- [9] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requir. Eng.*, vol. 11, no. 1, pp. 102–107, Nov. 2005.
- [10] V. R. Basili and H. D. Rombach, "The TAME project: towards improvement-oriented software environments," *IEEE Trans. Softw. Eng.*, vol. 14, no. 6, pp. 758–773, Jun. 1988.
- [11] J. Tian, J. Troster, and J. Palma, "Tool support for software measurement, analysis and improvement," *J. Syst. Softw.*, vol. 39, no. 2, pp. 165–178, Nov. 1997.
- [12] L. Lavazza, "Providing automated support for the GQM measurement process," *IEEE Softw.*, vol. 17, no. 3, pp. 56–62, 2000.
- [13] S. Komi-Sirvio, P. Parviainen, and J. Ronkainen, "Measurement automation: methodological background and practical solutions a multiple case study," in *Proc. of the 7th International Software Metrics Symposium*, 2001, pp. 306–316.
- [14] S. Chulani, B. Ray, P. Santhanam, and R. Leszkowicz, "Metrics for managing customer view of software quality," in *Proc. of the 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717)*, 2003, pp. 189–198.
- [15] G. de S. P. Moreira, R. P. Mellado, D. A. Montini, L. A. V. Dias, and A. M. da Cunha, "Software product measurement and analysis in a continuous integration environment," in *Proc. of the 7th International Conference on Information Technology: New Generations*, 2010, pp. 1177–1182.
- [16] G. Ghezzi, H. C. Gall, "SOFAS: A Lightweight Architecture for Software Analysis as a Service," in *Proc. of the Ninth Working IEEE/IFIP Conference on Software Architecture*, 2011, pp. 93–102.
- [17] B. Caglayan, A. T. Misirli, G. Calikli, A. Bener, T. Aytac, and B. Turhan, "Dione: an integrated measurement and defect prediction solution," in *Proc. of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*, 2012, pp. 1–4.
- [18] M. Jureczko, J. Magott, "QualitySpy: a framework for monitoring software development processes," in *Journal of Theoretical and Applied Computer Science*, v. 6, n. 1, 2012, pp. 35–45.
- [19] A. Janus, R. Dumke, A. Schmietendorf, and J. Jager, "The 3C approach for agile quality assurance," in *Proc. of the 3rd International Workshop on Emerging Trends in Software Metrics (WETSOM)*, 2012, pp. 9–13.
- [20] B. Keser, T. Iyidogan, and B. Ozkan, "ASSIST: an integrated measurement tool," in *Proc. of the Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, 2013, pp. 237–242.
- [21] L. Madeyski, and M. Majchrzak, "Software measurement and defect prediction with DePress extensible framework". *Foundations of Computing and Decision Sciences*, v. 39, n. 4, 2014, p. 249–270.
- [22] CMMI-DEV, "Improving processes for better products and services," in *CMMI for Development, Version 1.3, CMU/SEI-2010-TR33*, SEI, Carnegie Mellon University, Pittsburgh, 2010.
- [23] V. R. Basili, H. D. Rombach, and G. Caldiera, "Goal Question Metric paradigm", *Encyclopedia of Software Engineering*, 2 Volume Set, John Wiley & Sons, Inc., 2004.
- [24] R. F. Calhau and R. A. Falbo, "An ontology-based approach for semantic integration," in *Proc. of the 14th IEEE International Enterprise Distributed Object Computing Conference*, 2010, pp. 111–120.
- [25] C. Liu, J. Wang, Y. Wen, and Y. Han, "A unified data and service integration approach for dynamic business collaboration," in *IEEE 1st International Conference on Services Economics*, 2012, pp. 54–61.
- [26] W. J. Yan, P. S. Tan, and E. W. Lee, "A web services-enabled B2B integration approach for SMEs," in *Proc. of the 6th IEEE International Conference on Industrial Informatics*, 2008, no. Indin, pp. 774–779.
- [27] J. C. Nardi, R. A. Falbo, and J. P. A. Almeida, "Foundational ontologies for semantic integration in EAI: a systematic literature review," in *I3E 2013, IFIP Advances in Information and Communication Technology*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 238–249.
- [28] S. V. Pokraev, "Model-driven Semantic Integration of Service-Oriented Applications," PhD thesis, University of Twente, 2009.
- [29] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "Using mapping studies as the basis for further research: a participant-observer case study," *Journal of Information and Software Technology*, 53, 2011, pp. 638–651.

Toward using Business Process Intelligence to Support Incident Management Metrics Selection and Service Improvement

Bianca Trinkenreich, Valdemar T. F. Confort, Gleison Santos, Flávia Maria Santoro

Department of Computing

Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

Rio de Janeiro, RJ, Brazil

{bianca.trinkenreich, valdemar.confort, gleison.santos, flavia.santoro}@uniriotec.br

Abstract— Background: Businesses are increasingly dependent on IT services, and providers need to deliver fast, with high quality and low cost. An incident is an event that can lead to loss or disruption of services. Incident management reinstates normal service operation as quickly as possible and mitigates negative impact to business, ensuring agreed levels of service quality. So, reduce resolution time is usually the most important goal for incidents. **Aims** We aim to obtain knowledge about process and identify adequate metrics for Incident Management to help reduction of resolution time. Our research questions are: (i) Which Incident Management sub-process is causing more impact to resolution time? (ii) Which metrics can be used to measure this sub-process? (iii) What actions can be taken to improve Incident Management process in order to reduce impact of this sub-process in resolution time? **Method:** We present a case study in a global large company that considers reduction of incidents resolution time as a goal. **Results:** By applying BPM, BPI and Process Mining we were able to discover the underlined process and a bottleneck for resolution time. Moreover, we proposed metrics to improve process and service quality by applying GQM+Strategies.

Keywords: Measurement, Metric, Incident Management, BPI, Process Mining

I. INTRODUCTION

Service is about delivering value to customers by facilitating results they want to achieve without taking costs and risk ownership. Many organizations started to outsource IT in order to build services capabilities to its products, being able to provide faster and more accurate service to customers [22].

IT service management is a set of specialized organizational capabilities for providing value to customers through services. Its practice has been growing by adopting an IT management service-oriented approach to support applications, infrastructure and processes [10]. In software engineering approach, while software development delivers a product, software maintenance provides service to customers, as it modifies a software system or component after delivery to correct faults, improves performance or other attributes, or

adapts to a changed environment. Accepting software maintenance as a service perspective, it should follow IT service management and be developed and improved as a service either [23].

Guidance on how to develop and improve IT service maturity practices is a key factor to improve service performance and customer satisfaction [8]. CMMI-SVC (Forrester et al., 2010) model was created to attend this need and it is based on traditional models like ITIL [10] and international standards as ISO/IEC 20000 [11]. This model requires the identification of appropriate metrics in order to monitor the various processes executed for service delivering to customers. Business Process Management (BPM) [1] is also used to support processes improvement and governance initiatives. BPM recognizes Information Technology (IT) as one of its pillars [1] and has broad application in business areas. We perceive convergence between both areas. However, references of BPM techniques supporting CMMI-SVC are not common. Having an adequate metrics selection is one of key success for measurement initiatives. In BPM, Business Process Intelligence (BPI) and Process Mining [4] provide solutions on how to support knowledge management of a business process. Furthermore, we can evaluate a process, verify which sub-process is causing more impact to process results and also to support selection of metrics associated to this sub-process, using BPI techniques.

We had selected Incident Management process inspired by Business Process Intelligence Challenge 2013 and 2014 [7], recognizing that it produces a large amount of log data that can drive knowing users' needs and issues, being a process that is also used for corrective software maintenance support, a process that is mostly considered by service providers for measurement initiatives [20], and the one that, when properly used as a strategic asset [15], can help to provide answers about the service quality [8]. Like that, we are also converging two major fields: Quality of Service and IT Service Maturity Models, and BPM and Process Mining.

This paper presents an approach to support Measurement and Maturity Models on selection of Incident Management metrics to attend organization goals by using BPM, BPI and

(DOI reference number: 10.18293/SEKE2015-110)

Process Mining. Besides this introduction, in Section 2 we present a theoretical framework, in Section 3 we present related works, in Section 4 we present the case study and finally our final considerations in Section 5.

II. 2 THEORETICAL FRAMEWORK

A. *IT Service Quality, Maturity Models and Measurement*

A service is an intangible and non-storable deliver customer value way, making it easier to get results without needing to be responsible for risks and costs associated with the work [8] [9]. Guidance on how to develop and improve service maturity is a key factor for service provider performance and customer satisfaction. In order to be able to offer quality, the supplier must continually assess the way service is being provided and what the customer expects. A customer will be unsatisfied with IT service providers who occasionally exceed expectations but at other times disappoint. Providing consistent quality is important, but is also one of the most difficult aspects of software and service industry [11].

Maturity models focus on improving organizations processes due to the assumption that product or system quality is highly influenced by the quality of process used to develop and keep it. Through essential elements of effective processes and an evolutionary path for improvement, maturity models provide guidelines on how to design processes, as an application of principles to meet the endless cycle of process improvement [8].

CMMI-SVC [8] is a service maturity model based on CMMI concepts and practices and other standards and service models, such as ITIL [10], COBIT [12], ISO/IEC 20000 [11], and ITSCMM [13]. CMMI-SVC is composed of 24 process areas and 5 maturity levels (from 2 to 5), and includes required activities to create, deliver and manage services [8]. Maturity levels are used as the obtained classification in assessments, and also as a recommended evolutionary path for organizations intending to improve their processes in order to provide better services. Although CMMI-SVC is not intended to be used only by IT firms, those organizations certainly can benefit of it. Measurement is included since CMMI initial levels. Due to the financial difficult to measure and control all existing processes, selected measures must be aligned to organizational business goals in order to provide effective information for decision making. This choice is not trivial [9]. GQM paradigm (Goal Question Metric) is generally used to select metrics to provide expected visibility of the organization's strategic objectives. Some criteria for selecting metrics are: relationships with strategic objectives, coverage of lifelong service, availability and objectivity frequency that data can be collected, changing sensitivity, performance visibility and representation [8].

GQM+Strategies approach [21] is an extension of the GQM, and takes strategies and KPIs as input for a model from the business level down to project and operational levels and back up. GQM+Strategies focuses on filling organizations vertical gaps and help creation of objectives and strategies and deriving metrics that are aligned with high-level business goals and also providing a mechanism to monitor success and failure of goals and strategies through measurement.

This approach has two core components: a process and a model, and introduces the idea of having multi-level goals, strategies, context, assumptions, and a broader interpretation model. The process includes generating a grid that represents the hierarchy of operational, strategies, measures and interpretations models, planning, executing and evaluating those strategies with recommendations for future improvements. The model is called "grid", which is a comprehensive model with a notation to support organizations on developing their operational and measureable business goals, selecting strategies to implement them, providing communication about those goals and strategies to stakeholders of organization, and then deriving those goals into aligned sets of other lower-level goals and strategies. Through the grid, all parts of the organization can recognize their role on reaching top level goals, how it can be measured and how results are interpreted [21].

GQM+Strategies approach basic concepts are about Organization goals (what organization wants to achieve to reach its objectives), Strategies (how to achieve those goals), Context Factors (external and internal environments), Assumptions (unknown estimations) and GQM graphs (how to measure if goal was reached and strategy was successful or failed) [21].

B. *Incident Management*

According to ITIL [10], an incident is defined as an unplanned interruption or quality reduction while providing IT services. For example, a failure or hardware configuration that has not yet impacted software or services is also considered incident. Incident Management deals with all types of incidents, including failures or questions made by users or automatically detected by monitoring tools.

Incidents are events that, if not addressed, can eventually cause the supplier to fail in meeting its service level agreement (SLA) [8]. SLA is an agreement between service provider and customer, describing the service that will be provided, documenting service levels goals, and each involved part responsibilities. A SLA failure can cause contractual penalties to be applied [4]. Service Desk is the first support level, the team that receives users requests and process according defined flow, providing solution using knowledge articles information, or when is not possible to solve the issue or attend the request, assigning incident ticket for next levels support.

In CMMI-SVC [8], Incident Resolution and Prevention (IRP) is the level 3 process area that handles incidents solution and prevention through problems analyzing and treatment. IRP activities include: identify, analyze, assign, monitor status and progress, escalate if needed, and implement workarounds to restore service, communicate progress, validate resolution with stakeholders and analyze root cause of incidents.

Avoiding overflow within the SLA incident response is usually one of the strategic goals of service providers who pay penalty for noncompliance. Knowing the behavior patterns of Incident Management process allows the identification of which sub-processes are not performing well and needs to be controlled and improved.

C. Process Intelligence

Business Process Intelligence (BPI) refers to the application of Business Intelligence techniques (BI) to business processes [2] [3]. Typically, data sources come from Business Process Management Systems (BPMS) and Process-Aware of Information Systems (PAIS). Recording business events that occur during the execution of processes are called Event Logs.

Event log reveals important insights on how a process actually works. Here we look at more specific BPI role, identifying three major scenarios for application event log usage: automatic process discovery, performance analysis and verification of conformity [1]. The first is about the way process is implemented in practice; the second provides a discussion about operation cost and time and the third is to verify if a set of rules is being followed. Process mining is an enabling technology for CPM (Corporate Performance Management), BPI (Business Process Intelligence), TQM (Total Quality Management), Six Sigma and others [4]. However, mining processes from an event log is not a trivial task. A processes mining project can be developed in five stages [4]. Figure 1 reproduces this cycle.

In stage 0, project is justified and planned. In stage 1, event data, models, objectives and questions are extracted. At this point, a domain expert presence is important to define issues and assist in data understanding. Stage 2 takes as input the output of previous stage and focuses on producing an event log and a control flow model connected to this event log. In this phase, process discovery techniques can be applied. With a relatively structured process, other perspectives (date, time, resource) can be evaluated during stage 3. As a result of step 3, a process model will be discovered and an event log will be used to provide operational support. In stage 4, knowledge extracted from previous stage is combined with running cases, providing chance for interventions, predictions and recommendations. We stand out that stages 3 and 4 can only be achieved if the process is sufficiently stable and structured [4].

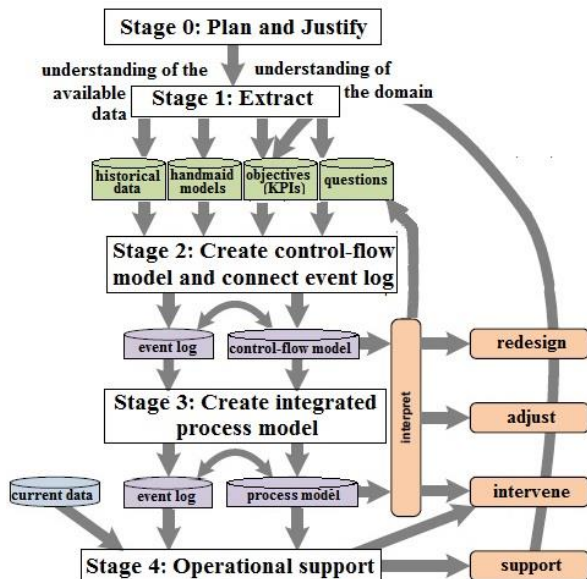


Figure 1 - Life cycle showing a mining process project stages [4].

III. RELATED WORKS

The scope of this paper is applying BPI techniques to a real incidents activities log in order to discover the Incident Management process and select its sub-processes that are more influencing results. This differs importantly from the field of root cause analysis because we are not investigating why incidents are occurring, but how long they take to be solved and which activity is causing more impact and could further be analyzed in order to improve incidents solving time. Nevertheless, much of our related work comes from incidents analysis and repairing. Following paragraphs show how our investigation and exploratory study had been inspired by methods and results from this area, even though no previous work precisely shares our scope.

Franke et al. [16] use incidents logs from 1.800 incidents in a large Nordic bank to find statistical distribution of IT service time to recovery. Authors show that log-normal distribution can help to understand what the best fit of IT service time to recovery, predicting downtime and costs to be used by organizations would be. This article does not use activities log to find what sub-process is taking more time to happen and impacting total process results and does not use Business process intelligence (BPI) for analysis neither.

Both 2013 and 2014 International Business Process Intelligence Challenges were about analyzing logs activities through process mining [7]. 2013 edition was about Incidents and Problems and 2014 edition was about finding correlation between Changes and Incidents.

Ferreira and Rabuge [17] had presented a methodology to apply business process analysis in a hospital emergency service to identify regular behavior, process variants, and exceptional medical cases. Ferreira and Silva [27] had presented another case study to determine how far the current incident management process is from the best practices described by ITIL in order to draw requirements for the new system. Although they also use process mining to extract the behavior of the existing process, they do not use it to discover sub-processes and which part is impacting the most the results in order to support measures selections for IT services maturity model. Also, they had used ProM tool [5], we had used Disco tool [14].

We could also notice a growing body of knowledge about case studies in different application domains regarding to describe reverse engineering with process mining from event logs. Mieke et al. [18] had analyzed procurement process to understand about internal fraud risk reduction.

IV. CASE STUDY

A case study method is an exploratory research technique used to highlight and explore aspects, which may guide providing directions for the question. This method is relevant for information system when researcher can study it in a real environment, and allows answering “how” and “why” questions. Although this paper scope is measurement, which is a quantitatively approach, a case study is commonly used as a qualitative method for researching information systems [19]. The research process we had used is explained as follows.

A. Case Study Planning

In order to execute the case study, we followed a set of stages depicted in Figure 2. Research questions that we aim to answer are: (i) Which Incident Management sub-process is causing more impact to resolution time? (ii) Which metrics can be used to measure and control this sub-process? (iii) What actions can be taken to improve Incident Management process in order to reduce impact of this sub-process in resolution time?

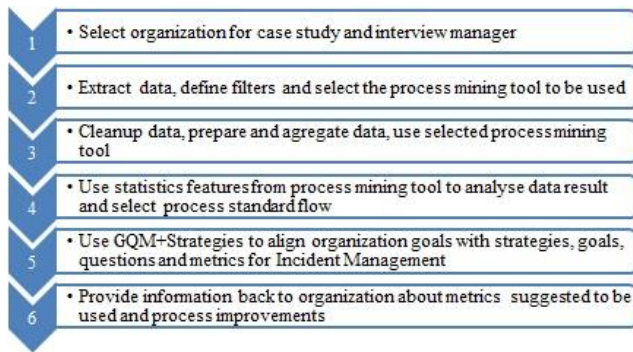


Figure 2 - Case study stages

B. Case Study Execution

In first stage, we identified an organization to perform the case study: Organization A is a large global organization headquartered in Brazil. It operates in over 30 countries and has offices, operations, exploration and joint ventures across five continents. The case study was performed on its IT services infrastructure department. One of the researchers that conducted the case study works there on improving quality of services and other researcher is a BPM and BPI specialist.

The IT Services Department provide IT services for all other departments of the organization following ITIL library practices [10], but it is not certified by any software or services maturity model. Incident Manager was interviewed and informed that he spends lots of effort to perform, analyze, report and plan new metrics in order to attend organizational goal of reducing incidents resolution time. Also, he had mentioned that Organization A uses only three Incident Management metrics, that indirectly should support attending resolution time: “Incidents resolved within target” (number of closed incident requests that were in accordance to service level agreement time), “Service desk resolutions” (number of incidents that were both registered and resolved by first level support without the assistance from another team) and “Incidents backlog” (number of not solved incidents). However, he pointed out that only these metrics are not effective and enough to provide results of reducing resolution time.

We had initiated the second stage by extracting and validating data availability through Organization A Incident Management tool (HP Service Manager 9 – SM9). Our main challenge was to understand SM9 format of event log data. Our effort was reduced by having help from Incident Manager. We had performed a preliminary evaluation to check if extracted data was compatible with defined process mining tools selected to use. Two tools were considered eligible as able to transform activity history on default event log XES [6]: ProM [5] and

Disco [14]. We started using both tools in order to select the most appropriated for considered scenario. We decided to use data for one month (April 2014), due to the huge amount of data, and one application (Intranet), because Incident Manager reported that it is the application that concentrates larger amount of incidents being opened by users.

Because there was no preconceived design process, we conducted a first execution to discover Incident Management process itself and answer some basic questions about it: how many incidents were opened in a determined month, what are the minimum, maximum and mean time closing an incident and which possible flows an incident resolution can have.

ProM tool [5] could not recognize control flow with some instances and this was the first issue found about it. Disco tool [14] could discover control flow without removals or adjustments to original data. Also, Disco tool abstracts its algorithm for process detection, and it resulted in lower effort. Because of those two difficulties about ProM tool, we had chosen to continue with Disco tool.

In third stage we had filtered incidents selected period and application, and started to prepare data. One incident has many activities and events, representing incident lifecycle. Original log file had 14.815 events. From those, there were 120 different activities. After extraction, we had to perform some cleanup of wrong entries that were not representing real activity names, for example concatenating an activity name with an incident number in same field. Possibly it was because some kind of bug in SM9 extraction tool, but as they represented only 1% of total, we removed and not considered entire incident registry for those. So, we had excluded 1% of incidents from data because of what we had considered bad data. After removing them by using Disco tool [14] functionality to filter undesired activities, we could get a process with 27 activities. Through analysis of process variants, we found 507 different paths, from 993 total identified process instances. Although it answers one of basic questions (which flows can an incident resolution has?), at this point this high number of activities and transitions did not allowed us to answer research question (i).

The control flow model identified by Disco tool [14] at this point had a lot of activities and transitions. At this stage, we used a Disco tool native functionality to aggregate some activities and transitions. For example, there are flow cases that go from activity A to activity B and then activity C. There are also cases where activity A flows directly to activity C. Disco tool does easily abstract these two types of cases making a control flow that considers only transition from A to C. Therefore, we can choose where to drill-down from a general and major flow to a detailed one.

For this research, we considered the most regular flow (with 5 steps) and used Disco statistics and performance features to analyze amount of elapsed time of each transition, in order to help us answering research question (i).

In fourth stage, we had used global statistics feature to identify the amount of total events (11,203 events) and answer basic questions: How many incidents were opened and what are the minimum, maximum and mean time closing an incident. Total of 993 incidents had been opened, with 5.5 days

of mean and 4.7 days of median to be solved. Minimum time to close an incident was 22 minutes and maximum had lasted 36 days. Even though, as we have shown that most cases had been solved with only 5 activities (Figure 3), this was considered a standard for analysis. Based on this, we had simplified incidents flow to a 5-steps process: Open, Assignment, Start Work in Progress, Resolved and Closed.



Figure 3 - Graph from Disco tool [14]

Table 1 provides time performance analysis for each transition of the considered 5-steps process. We can notice that it is taking more for someone to take responsibility to solve the incident (Open to Assessment) than to properly solve it (Work in Progress to Resolved), answering research question (i) “Which Incident Management sub-process is causing more impact to resolution time?”

TABLE I. DURATION ANALYSIS FOR EACH TRANSITION

Transition	Total Duration	Max Duration	Mean Duration	Median Duration
Open → Assignment	68,6 days	26,3 hours	109,3 minutes	51,5 minutes
Assignment → Work in Progress	11,6 days	5,9 days	24,6 minutes	68,5 seconds
Work in Progress → Resolved	13,5 days	22,4 hours	49,2 minutes	11,4 minutes
Resolved → Closed	111 months	4 days	3,8 days	4 days

In fifth stage we used GQM+Strategies [21] to align goals, strategies, questions and metrics (Figure 4) in order to suggest a measurement improvement for Organization A, and answer research question (ii) “Which metrics can be used to measure and control this sub-process?”. We have used root cause analysis, found that Assignment is taking much time because service desk commit assignment errors, taking longer to define correct team to send incident, which happens because lack of available, correct and updated information. Considered context factor for GQM+Strategies was cost reduction scenario that Organization A is facing. Considered assumptions for GQM+Strategies was there is already human resources available and with enough expertise to generate information and update knowledge articles. New metrics suggested were: “Time to Own” (number of minutes that an incident is taking to be assigned to correct team), “Incident Assignment Correctness” (percentage of incidents that were assigned to correct team) and “Articles not updated” (number of times that service desk cannot find required information to solve an incident or assign to correct team).

In sixth stage we proposed process improvements for

“Open to Assignment” part of Incident Management process. Knowledge articles used by first level support represent the way that service desk team is able to solve incidents by itself and also assign to proper higher support teams when cannot be solved in first level. So, improve these artifacts is a way to make first level capable of solving more incidents and also reduce time and errors in Assignment phase, which is the bottleneck for incident resolution time (and reducing is the organization goal). Knowledge articles should contain direct, simple and proper questions for a first level support analyst to do when a user calls reporting an error or requesting a service. Search mechanisms should provide easy finding of articles by many key words.

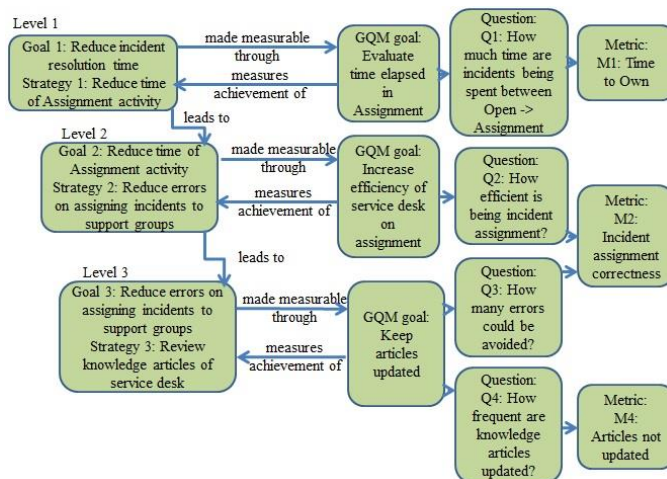


Figure 4 - GQM+Strategies diagram proposed for Organization A

C. Results and discussion

We could recognize that Incident Management is a business process as any other. Therefore, BPM lifecycle is able to assist in its improvement, since the process identification and its variations (according to criticality, for example) to its monitoring & control and redesign.

First result was discovery of actual Incident Management process by extracted data. From 5-steps process (Open, Assignment, Work in Progress, Resolved and Closed) we had taken statistics to measure time performance for each transition, and answered research question (i). Transition “Open to Assignment” was the bottleneck, the one causing more impact to resolution time. After mining incident event logs, we found that Organization A is taking, in average, more than double time to assign proper support team than to actually solve an incident. This shows also that stakeholder could discard transition “Resolved to Closed”, that is, this transition not critical as the majority of this transition was handled automatically.

Incident Manager for Organization A had explained us that first level support uses knowledge articles to understand what is being requested by user and for what team should be assigned to solve it. He had also informed that many times there is no information, or information in not updated about support teams for each service, and because of that, first level support can commit assignment errors. Then we built a

GQM+Strategies grid to organize and propose strategies and derive in questions, goals and metrics to be used by Organizational A in order to have a better control of Incident Management process and take proper actions to reach organization goal (Figure 4).

Selection of most appropriate metrics to be used is not a trivial task and is the success key for effective control and adherence to quality and process performance goals. This approach can support metrics selection activity required for Measurement and Analysis (CMMI-SVC level 3). By only using incident activities logs and interviewing Incident manager, we were able to (i) discover all possible process flows, (ii) find the process flow that is more commonly followed, (iii) find transition time between steps that is taking longer, and (iv) derive organization goal in strategies and metrics to improve quality of service.

V. CONCLUSION AND FUTURE WORK

In this paper we aimed to use BPM, BPI and Process Mining to discover Incident Management process by event logs and find bottleneck for incident resolution time, in order to support selection of metrics to attend organization goals and quality of services improvement. We identified areas that allow us to collaborate and contribute on a mutual basis Business Process Management and Quality of Services. Our objective was to link the theory and practical approach to generate knowledge and enable decision-making in one of these areas using Business Process Intelligence and IT Services Maturity Levels. The result would be the recommendations for the appropriate service and capacity model to support mining in IT operations. Through this collaboration, we expect to improve resources to support this initiative. We had obtained evidences that Business Process Management (BPM) and Business Process Intelligence (BPI) can be combined to support services maturity models. Also, we point there is little discussion about this aspect and that proposed approach can be conveniently and efficiently applied.

This case study can be modeled as a new method to be reproduced in order to select metrics by process bottleneck discovery and support root cause analysis for problems. Also, it shows we can generate knowledge about the process to support quality of services.

We can separate future work in two phases. First phase involves investigating methodological aspects: what, how and where to apply BPM and BPI techniques in service processes areas. Second phase involves researches that can help to quantify and to qualify the application of first phase in real situations. This will encompass analysis of case studies and interviews with experts. Dynamics within two phases could identify trends to improve IT solutions application within organizations and reduce required time for an organization to achieve a measurement process of IT services maturity models as well as the cost to maintain this level.

This is the first step we have taken in a path we argue is important to be followed. We hope to debate and envision new opportunities for action and believe the two areas converge from an academic and industrial point of view. The result will be the ability of increasing technological information efficiency

to provide better solutions to organizations and society.

ACKNOWLEDGMENT

Authors would like to thank the financial support granted by FAPERJ (project E-26/110.438/2014).

REFERENCES

- [1] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A. *Fundamentals of Business Process Management*, Springer, Berlin, Germany (2013)
- [2] Castellanos, M., K.A.d. Medeiros, J. Mendling, B. Weber, and A.J.M.M. Weitjers, *Business Process Intelligence*, in *Handbook of Research on Business Process Modeling*, J. J. Cardoso and W.M.P. van der Aalst, Editors. 2009, Idea Group Inc. p. 456-480 (2009)
- [3] Grigori, D., Casati, F., Catellanos, M, Dayal, U., Sayal, M., & Shan, M. *Business Process Intelligence*. *Computer in Industry*, 53(3),321-343.(2004)
- [4] Aalst, W.M.P., van der, et al.: *Process Mining Manifesto*. Technical Report, IEEE Task Force, (2011)
- [5] *Process Mining Framework*, <http://www.processmining.org/prom/start> (2014)
- [6] *Extensible Events Stream*, <http://www.xes-standard.org/> (2012)
- [7] *Ceurs Workshop Proceedings, Business Process Intelligence Challenge 2013*, <http://ceur-ws.org/Vol-1052/> (2013)
- [8] CMMI INSTITUTE. "CMMI-SVC – Capability Maturity Model Integration for Services". <http://cmmiinstitute.com/resource/cmmi-for-services-version-1-3/> (2010)
- [9] Florac, W. A., Carleton, A. D., *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison Wesley (1999)
- [10] *ITIL Version 3 Service Design*. Office of Government Commerce (OGC) (2007)
- [11] ISO - International Standard Organization "ISO/IEC 20.000-2: Information Technology – Service Management – Part 2: Code of practice" - Switzerland. (2012)
- [12] *Information Systems Audit, et al. COBIT Five: A Business Framework for the Governance and Management of Enterprise IT*. ISACA (2012)
- [13] Niessink, F., Clerc, V., Tjldink, T., Vliet, H. *The IT Service Capability Maturity Model - IT Service CMM, version 1.0RC1*. (2005)
- [14] *Process Mining and Automated Process Discovery Software for Professionals - Fluxicon Disco*, <http://fluxicon.com/disco/> (2014)
- [15] Solingen, R., Berghout, E. "The Goal Question Metric Method: A Practical Guide for Quality Improvement of Software Development" McGraw-Hill (1999)
- [16] Franke, U., Holm, H., Konig, J., *The Distribution of Time to Recovery of Enterprise IT Services - IEEE Transactions on Reliability* (2014)
- [17] Ferreira, D., Silva, M. *Using Process Mining for ITIL Assessment: A Case Study with Incident Management*. *Proceedings of 13th Annual UKAIS Conference* (2008)
- [18] Mieke, J.; Lybaert, N.; Vanhoof, K.: *Business Process Mining for Internal Fraud Risk Reduction: Results of a Case Study*. *Induction of Process Models* (2008)
- [19] Recker, J. "Scientific Research in Information Systems A Beginner's Guide". Springer ISBN 978-3-642-30048-6 (2013)
- [20] Trinkenreich, B., Santos, G., Barcellos, M.. "Metrics to Support IT Service Maturity Models – A Systematic Mapping Study", 17th International Conference on Enterprise Information Systems (ICEIS), Barcelona, Spain (2015)
- [21] Basili, V., Trendowicz, A., Kowalczyk, M., Heidrich, J., Seaman, C., Münch, J., Rombach D., "Aligning Organizations Through Measurement - The GQM+Strategies Approach". Springer (2005)
- [22] Davenport, T. "Process Innovation: Reengineering Work Through Information Technology". Harvard Business Press (2013)
- [23] Niessink, F., Vliet, H., "Software Maintenance from a Service Perspective". *Journal of Software Maintenance*. v12 p.103-120 (2000)

Study on the Accident-causing Model Based on Safety Region and Applications in China Railway Transportation System

Qin yong Ma hui Jia limin
State Key Lab of Rail Traffic Control & Safety
Beijing Jiaotong University
Beijing, China
yqin@bjtu.edu.cn

Ma hui Du Miao
School of Traffic and Transportation
Beijing Jiaotong University
Beijing, China

Du miao
Tianjin metro operation company
Tianjin metro operation company
Tianjin, China

Abstract—In order to quantitatively and systematically explain the accident occur process and assess the risk for the complex system, this paper proposes a new accident-causing analysis model, i.e. perturbation-safety region (P-SR) model. In this model, the safety region definition is introduced for the quantitative description of the system safe status; also the change process of the system risk is analyzed. The four relative parts included in this model are described in details, such as the risk resource part, the perturbation part, the alarm and system change part, and the accident part. Finally, the proposed model is applied to railway transportation system, and the Wenzhou train collision is systematically analyzed, also the specified control measure for the train emergency dispatch is demonstrated.

Keywords—*accident-causing analysis ;complex system; perturbation; safety region; railway transportation system;*

I. INTRODUCTION

Accident-causing theory mainly studies why accident happens and the mechanism of its process [1]. In order to prevent future accidents, the relationship of the causation found out in each part of procedure is established by disclosing the interaction of the components in the system. Traditional accident-causing theory, like Domino theory proposed by Heinrich in the 1940s [2], takes single element such as human, equipment or other causes separately into consideration as a chain or sequence of events [3], which explains well accidents caused by physical components and relatively simple systems [4]. Whilst systems we build today are increasingly complex that linear model is no longer adequate to capture the interactions and coupling within the system; thus it requires us to analyze the accident causation systematically as a whole. To catch up with the complexity, the accident theories developed via previous linear causation theories to present-day systematic theories, such as: system theory, perturbation accident-causing theory, energy transfer theory and information theory [5].

The systems approach addresses the notion that safety is an emergent property, which arises from non-linear interactions between multiple components across complex system and the relationship of behaviors implicated in operation [6]. In systemic safety models, the accident process is described as a complex and interconnected network of events to model the dynamics of complex systems [7]. Rasmussen's hierarchical sociotechnical framework [8] and Leveson's system theoretic accident modeling and processes [4] are two notable approaches. Even though these accident models considered the joint effect of multi-factors in an accident with their dynamic interactions, the descriptions of them (human, equipment, environment and etc.) are mainly qualitative, and the outcome of those interactions of system components are described respectively without an uniform expression. On the one hand, these models are sufficient to help us learn from accidents that have already happened, and thereby preventing hazards from the similar kind. On the other, as they hardly reveal the course of the outcome of system change, they are inadequate to guide real-time emergency response to prevent accident when the system is disturbed and prone to accident. This is mainly because the consideration of system state as a whole is lacked of in these models. And the challenges we meet today to achieve safety is going beyond accident analysis to the extent of resilience engineering [9]. Hereby, the accident analysis should also be able to implement in the real-time field work to prevent accident not only after but during its process, by enhancing its resilience against disturbance.

To achieve this goal, the conception of safety region, which depict the safe state affected by different factors in a unified way, is introduced with the combination of perturbation accident-causing theory to establish the perturbation-safety region (P-SR) accident-causing theory. In this theory, in addition to analysis causality systemically, the safe state of the system after perturbation is described quantitatively with the changing course of it in P-SR model. And then by exploiting

the safe state as risk assessment, the monitoring and evaluation of system safe state as well as the corresponding control measures are brought into the model to enable its practicability in safety management of production activities.

In this paper, the P-SR accident theory is illustrated in section II, with detailed description of safety region and accident causing process. And then the application of the model is presented in section III: (1) the whole accident course and the crucial safety factors of railway system is extracted by the reconstruction of Wenzhou train collision; and (2) a specific emergency railway safe state restoration method-train rescheduling- follows to illustrate the process of how the safety control measure works in real operation.

II. THE ACCIDENT-CAUSING MODEL OF P-SR

Inevitable as perturbation is in production activities, Amalberti [10] argued that these ‘noises’ (e.g. equipment malfunction or human errors) jeopardize operation safety; conceptually they should be symmetrically assessed and then calculate the associated risks. With new safety methods and perspectives that keep up with the continuously increasing complexity of industry, accident models aiming at explaining events and guide risk assessment need to match this complexity [11]. Specific to the complex system, the P-SR model promotes a quantitative description of the safe state and risk boundary of the system, which will better instruct safety monitoring and relative control measures. The concept, perspectives and processes are defined and described in this section.

A. The Definition of Safety Region

Safety region analysis have been applied to monitor the safety and stability of power system [12]. The concept of region quantitatively describes the safety boundary of a system so that it could dynamically and consecutively monitor the system state with its changing process, and evaluate the safe state to provide warning information.

On the basis of the object studied in accident models, the safety region is defined as a changing space to describe the multifactor. Let $X = \{x_1, x_2, \dots, x_n\}$ be the set of characteristic variables representing the characteristic state of the system, in which n is the number of the critical subsystem. The characteristic variables, derived from multifactor of human, equipment, environment, management or other factors, contain both discrete variables and continuous variables. Define space E as safety region: within the boundary of E is safe space; otherwise is accident space \bar{E} . The boundary is determined by the threshold of system safe state, i.e. the accepted risk level that can ensure system safety.

The safety region is determined as a n dimension space by the number of the characteristic variables n , in which the lower dimension spatial scope may vary with high dimension variables. Fig.1 gives an example of a 3-dimension safety region composed of $X = \{x_1, x_2, x_3\}$, in which x_3 is a discrete variable, representing two types of system state at this dimension: when $x_3 = 0$, the safety region is E_0 ; when $x_3 = 1$, it changes to E_1 .

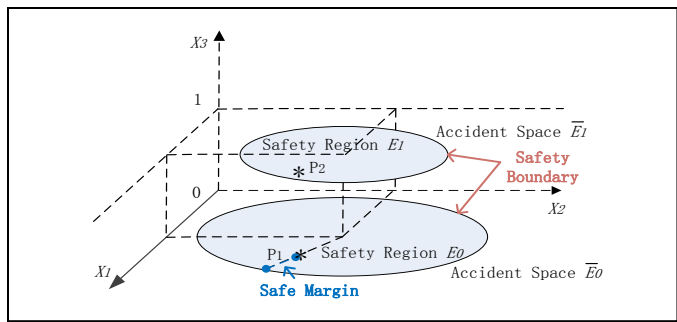


Figure 1. The change of system safety region

The boundary of the safety region is only determined specifically to a certain system. Usually, the state of the system located in safety region is called the balanced state. If the character point falls in the safe space, then the system is confirmed to be safe, with the distance between the point and boundary, called safe margin, to assess the safety level of the system. Otherwise, the point falls in the accident space when it breaks through the safety boundary, indicating that the safe state reaches an unacceptable level and then causes the accident.

In production activities, the system state continually deviates from safe space under the influence of perturbation. As it reaches a certain extent that beyond the safety boundary, the system enters the accident space. Fig. 2 show a safety region consists of 2 dimension variables $\{x_1, x_2\}$, in which P_1 and P_2 represent respectively system running safely and accident taking place. Obviously, the crucial task to use safety region to denote system safety is to obtain the safety boundary- a decision function returning a safe threshold that differentiate the state of safety and accident [13].

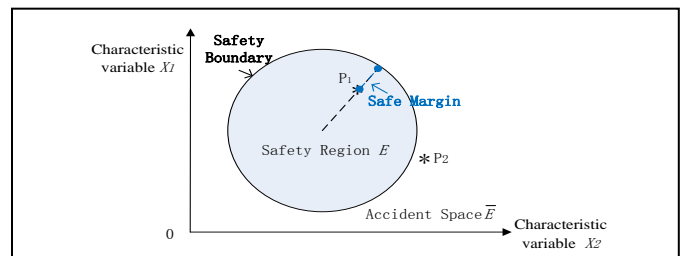


Figure 2. A schematic diagram of two-dimension safety region

B. The Analysis of Accident-causing Model Based on Perturbation-Safety Region

The P-SR accident-causing model (Fig.3) consists of four critical parts: the risk resource part, the perturbation part, the alarm and system change part, and the accident part.

To study the nature of accidents, in the first part, the risk resource is prominently analyzed in the perspective of energy carrier, followed by the analysis of the direct cause of perturbation. The moving device, electrified equipment, and containers loaded of hazardous chemicals constitute the energy carrier in the system, which is the material basis of an accident. And the severity of the accident is related to the types, quantity, property, status, and energy storage method of the energy carrier. Normally, the system maintains safety by effectively taking control of the energy. Only when the unsafe multifactor

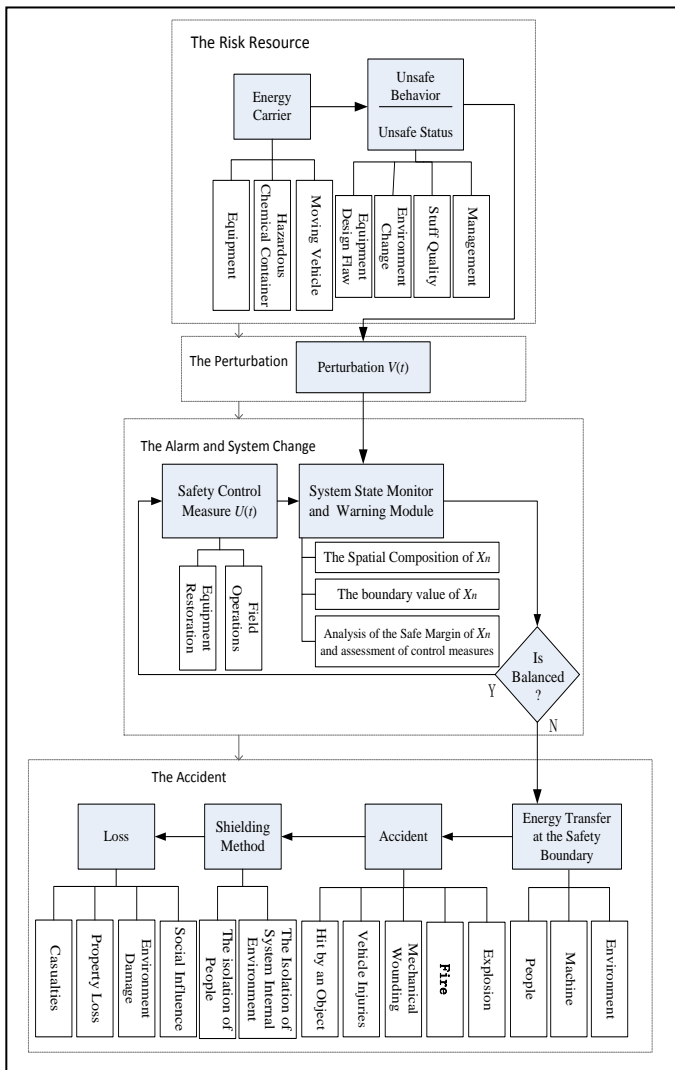


Figure 3. The Perturbation-Safety Region accident-causing model

disturbs the system will it result in failure of energy control—mainly because of the unsafe status and unsafe behavior:

- Unsafe status includes environment change and the defect of the equipment itself. First, natural disasters and extreme weather, e.g. lightning, earthquake, typhoon, debris flow and blizzard, are uncontrollable stochastic factors, which will influence the equipment and energy transmission in the system by causing the perturbation to the balanced state and further the accidental release of energy. Second, the equipment has problems of wear, deformation, and metal fatigue due to the long time use, thereby increasing the probability of mechanical fault. And the device itself may also have design flaws. Meanwhile, with the increasing complexity of the system, the dynamic interaction of each part is more complicated than the fault of single equipment may affect the whole system. Thus, the system is vulnerable to the unsafe state.
- Unsafe behavior mainly refers to the unsafe operation and management of human. The role people play in the

system mainly includes: design personnel, operation staff, maintenance staff and management personnel. They together determine the reliability, stability and safety of a system. Yet each person is an individual with different quality, characteristic, education and etc. In the process of production, man's operation ability, management level and experience are closely related to system safety. Unsafe behaviors such as sneaking off in work, illegal operation, the decision-making mistakes, and loose management are the possible causes of an accident.

The effect of the unsafe state and behavior engenders the perturbation $V(t)$, shown in the perturbation part of the model in Fig.3, which is the direct cause that deviate the safe state from balanced state. The perturbation should be further analyzed in term of the specific system and situations.

As the controllers or decision makers are highly dependent on feedbacks to take action after perturbation, the necessary information about the actual state of the process is crucial to avoid accidents [4]. The question then arises about how we express and present the actual safe state. In the next stage, the alarm and system change part, the concept of safety region we introduced is the solution to this problem. At the beginning, the initial balanced state is expressed as $X(t) = \{x_1(t), x_2(t), \dots, x_n(t) | x \in E\}$. After the perturbation, it changes to $X(t+1) = AX(t) + V(t), x \in E$, in which A is the system parameter. In order to ensure the system to still be in balanced after the disturbance, the changes of state in safety region need to be monitored so that the safe margin can be calculated. Then, according to the safe margin, corresponding prevention and control measures should be taken to rebalance the system. If the adopted measures are inadequate, the system will break the safety boundary and into the accident space. Herein, a system state monitor and warning module based on safety region is included in this part. As $X(t+1)$ moves to the safety boundary, the safe margin decreases; then the warning system generates alarm information; based on the alarm information, safety control measure $U(t)$ should be applied on the system, which is expressed as $X(t+1) = AX(t) + BU(t) + V(t), x \in E$ (B is the safety control parameter). If the system restores balance, it continues to monitor the change of safe margin and assess the control measures, so that the safety control measures module responds appropriately; if the system state broke the balanced state, it means undesired energy transfer has occurred and resulted in an accident.

Fig. 4 depicts the rebalance or accident procedure after perturbation under the action of system state monitor and early warning module (the arrows are the state locus, and the blue lines show the safe margin at each time).

The system is in balanced state before t_1 . At t_1 the safe state begins to move toward the safety boundary under the effect of perturbation $V(t)$. Then the warning module detects the reduction of the safe margin and raises alarm. Afterwards, the countermeasure $U(t)$ is applied at t_2 to slow down the decrease of safe margin. Later, the safe margin decreases slower at t_3 , indicating that the system tends to restore the balanced state. Still, appropriate safety measures continue to be implemented

at t_3 . Finally the safe margin begins to move toward the internal safe space at t_4 , which means the system state has been effectively controlled, thereby avoiding the accident.

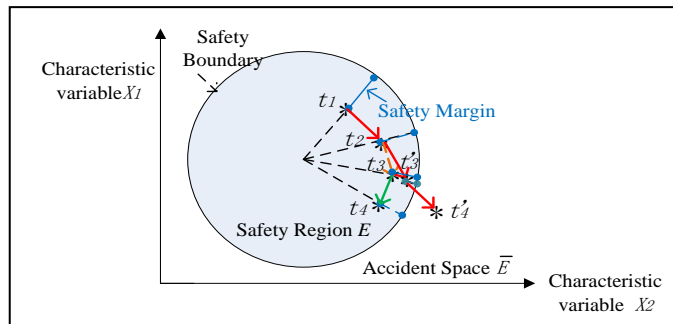


Figure 4. The trace of system state retrieving equilibrium state or heading to accident after perturbation

Another trace in Fig.4 shows an opposite situation where the safety measure $U(t)$ fails to work. The difference is that the countermeasure taken at t_2 is far enough to slow down the decreasing speed of the safe margin. Thus, at t_3 , the system state is already close to the safety boundary and keeps approaching it. Ultimately, the system state breaks through the boundary, with the energy (chemical energy, mechanical energy, kinetic energy, or electric energy) transferring to people, equipment, and environment.

According to the previous analysis, the safety control measure based on the monitor and warning module is critical to restore system safety after perturbation, as it decides the trend as well as the speed of the system state change. Therefore, in the accident prevention and control procedure, we should establish corresponding emergency plans specific to the object; and strengthen its disturbance control measures to reduce the probability of accidents, eventually avoiding the accidents.

Nevertheless, when the accident happens, there's still shielding method-the isolation of people, environment and energy carrier -we can take to control the damage degree of the energy release. If the shielding measure fails or not timely, the accident may cause severe direct loss like casualties and property loss, as well as the indirect loss such as damage of the environment, the social influence and the production stagnation, which is described in the accident part in Fig.3.

To sum up, the key of P-SR model is to extract the characteristic state variables of safety critical subsystem to build the safety region; and then determine the safety threshold to establish the safety boundary. That's when the system state can be quantitatively calculated as safe margin.

III. THE APPLICATION OF P-SR MODEL ON RAILWAY TRANSPORTATION SYSTEM

As China's railway transportation system thrives, the train speed is increasingly faster, train numbers are much denser, power supply capacity is bigger, and the multi-factors coupling is higher. With a lot of risk sources, the railway system is both an ultra-safe system and a typical complex system, confronted with enormous challenges of accident prevention and control. The P-SR model herein provides a solution to solve these

problems as the following one accident analysis example and one emergency control example confirm.

A. The Wenzhou Train collision Accident Analysis

According to the accident investigation report established by State Council of China [14], the P-SR model is employed to analyze and reconstruct the Wenzhou train collision process so as to provide decision support for future accident prevention and the improvement of safety measures.

On 23 July 2011, high speed train D301 from Beijing to FuZhou collided with the high speed train D3115 from Hangzhou to FuZhou on Yongwen railway line, Wenzhou, Zhejiang province, China. The analysis of the accident based on P-SR model is established in Table I.

As the relative speed and position of a train with adjacent trains is the essence to control safety, this system safety-critical state space is defined as three-dimensional: train running control mode, train speed and train interval. So the safety region is also three-dimension, in which the train running control mode is discrete variable with the value of automatic block control or manual control; the train speed is continuous variable ranging from 0~350 km/h; the train interval is discrete variable indicating the number of blocks between two trains running on the same rail at same direction. To facilitate the graphical display of the safety region, the traffic control mode is set as a third dimension, thus we can describe the changing of the system's safe state in two-dimensional space.

Previously we introduced that the special extent of the safety region in dimensionality reduction space is possible to vary with the value of high-dimension variables. In this example, along with the change of train running control mode, the boundary of the two-dimensional safety region made up by the train speed and train interval changes as well (see Fig.5). In automatic block control mode, also the normal operation mode, the safety space is in a large range as shown in area E_0 ; while in manual control mode, the spatial extent of safety region reduces to E_1 , as Automatic Train Protection (ATP) requires the speed to be lower than 20 km/h and the train interval is required to be as the distance between adjacent stations.

The system safety region composes of the velocity v (km/h) of the first train running onto the section and the interval of the subsequent train n (the number of the blocks between two successive trains). In automatic train control mode, the safety boundary is made up by the safety threshold of the train running speed of 250 km/h and the minimum safe interval of 2 blocks, as $E_0(v, n) = \{v \leq 250, n \geq 2\}$. In the manual mode, the safety threshold of the speed changes to 20 km/h and the minimum safety interval increases to 3 blocks, as $E_1(v, n) = \{v \leq 20, n \geq 3\}$, for sufficiently stopping the train before any collision.

The safety region is E_0 at t_1 , when D3115 set off from Yongjia station at a normal speed onto the section under automatic train control mode. However, the control mode changed into manual mode at t_2 , with the safety region narrowed down to E_1 . Soon after, D3115 was stopped by the ATP when running onto the track 5829AG with faulted track circuit. At the time of t_3 , D301 entered the same section

TABLE I. THE ACCIDENT-CAUSING ANALYSIS OF WENZHOU TRAIN COLLISION

Items		Content	
Energy Carrier		Moving motor train unit	
Trigger Factors	Unsafe Status	<ol style="list-style-type: none"> 1. The lightning activity was unusually intensive along the Wenzhou-Yongjia and Wenzhou-Ouhai railway line; 2. The host in the train control center only transfer the fault message received from track circuit to the monitor and maintenance terminal, while continuing outputting the signal control message according to the occupancy of track at the last moment before malfunction (the track was free so the control center authorized green signal). 3. The integrated wireless communication devices in D3115 lost its signal, so the driver couldn't connect to train dispatcher in time. 	
	Unsafe Behavior	Management	<ol style="list-style-type: none"> 1. The equipment design company had severe defects in the design process and quality control of control center equipment 2. The project director ministry had a series of management failures on equipment bidding, technical examination and inspection for service for newly developed signaling equipment 3. The field work company had loopholes and deficiencies in safety management and failed to adequately respond to equipment malfunction caused by lightning.
		Operation	<ol style="list-style-type: none"> 1. The field staff didn't perform joint interaction control of train running and track occupancy under manual mode. 2. The D315 was authorized onto the section at automatic control mode without confirmation that the D3115 had arrived at the next station or the equipment had restored to work normally.
The perturbation	<ol style="list-style-type: none"> 1. The lightning struck a trackside signal assembly, burning out its fuses F2, while the transmitter in track circuit 5829AG lost connection with the control center. 2. The control center gave an incorrect indication, based on the state before the fault when the track was free, that the track section containing train D3115 was unoccupied, thereby allowing the signal instruction staying green. 3. Due to the communication error between 5829AG track circuit and control center, 5829AG track circuit began to send messy code, causing the computer interlocking system in Wenzhou south station displayed red band on the corresponding section. 4. As D3115 run onto the malfunctioned track 5829AG, the messy code transmitted to the train triggered automatic braking of ATP, so that D3115 came to a halt with 3 times failure to override the system into visual driving mode. 		
The monitor and warning	<ol style="list-style-type: none"> 1. The computer interlocking system in Wenzhou south station appeared 'red band'. 2. The frequency shift track circuit terminal at mechanical room in Wenzhou south station displayed red alarm light 3. The last two communication boards in the track circuit interface unit in Wenzhou south station indicated red warning light. 4. The computer interlocking system in Wenzhou south station appeared 'red band', while the Centralized Traffic Control System (CTC) in dispatching station didn't. 		
Safety measures	<ol style="list-style-type: none"> 1. The track maintenance workers walked along the Wenzhou-Ouhai and Yongjia-Wenzhou railway line to check the occupancy of track. 2. The railway electricity workers attempted to restore the faulted equipment. 3. The train control mode was change from automatic control into manual control mode in Yongjia station, Wenzhou south station and Ouhai station. 4. The dispatcher instructed the driver of D3115 driving under visual mode at a speed lower than 20 km/h, when encountering red light in the section. 		
Accident Space			
Energy Transfer	Train D301 ran at 99 km/h crashed into the rear-end of the D3115 run at 16 km/h.		
Accident	The 15th and 16th coaches at the rear of D3115 and the front five coaches of D301 were derailed.		
Shielding	The driver of D301 pulled on emergency brake at the sight of D3115.		
Loss	40 people were killed and 172 injured; 7 motor train set vehicles was scrapped, 2 broken heavily, 5 broken at medium, 15 broken slightly; the network of Overhead Contact System in accident section collapsed; the railway line at accident section shut down for 32 hours and 35 minutes.		

TABLE II. THE MONITOR AND WARNING INFORMATION IN WENZHOU COLLISION AND CORRESPONDING EVALUATION

Time	The monitor and warning of equilibrium state	Safety measures	Safety Region	Safety Margin	Evaluation of safety measures
t_1	None	None	E_0	equilibrium state	—
t_2	The inconformity of the display in CTC and train control center	The train control mode was change to manual control mode in Yongjia station, Wenzhou south station .	E_1	Increasing	Effective
	Track circuit sent messy code	D3115 was stopped by the Automatic Train Protection (ATP)	E_1	Increasing	Effective
	None	The driver of train D3115 overrode the ATP and drove at visual mode.	E_1	Decreasing	Failed
t_3	None	The following train D301 approached onto the section of track where D3115 had been stopped at automatic mode	E_1	Decreasing dramatically	Dangerous
t_4	None	Emergency brake of D301	E_1	Enter accident space	Slight

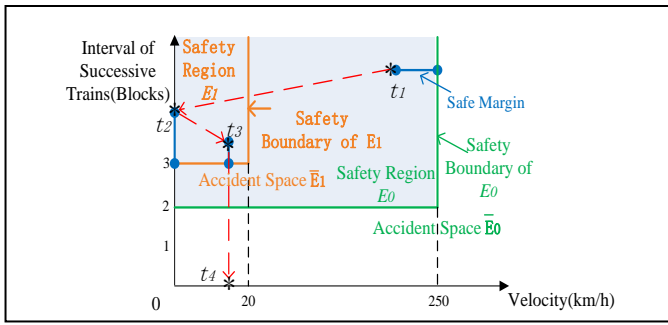


Figure 5. The evolution of system state in Wenzhou train collision based on safety region

occupied by D3115 as a way of the automatic mode, which it shouldn't. Two minutes later, D3115 finally overrode the ATP to start the visual driving mode. Nonetheless, the interval between these two trains decreased sharply at this time. As there was no effective warning, no imperative safety measure was taken. Thus the safe margin diminished dramatically. Eventually, D301 collided with D3115 at t_4 that the system state broke through the safety boundary, with energy transfer, causing the accident. The course of the accident is shown in Fig.5 as red arrow lines. The warning and monitor information with relative safety measures at each time is evaluated according to safe margin in Table II.

According to the analysis of the P-SR accident-causing model, it is the joint efforts and the interaction between multiple factors that put the system at risk of accident. However, it is the control measures that finally decide whether an accident will happen or not. In Wenzhou train collision accident, the safety measures adopted according to the early warning has somewhat maintained system safe margin. But when the system neither obtained the early warning information in the field, nor did any imperative human or equipment safety control measures are taken, the system safe margin began to drop dramatically until the accident happened.

B. Specified Application of the Safety Control Measures

This section focuses on the system safety control measures to restore the order of the system. Specific to the railway system, train dispatching and rescheduling is the imperative method to ensure both the operation safety and transportation capability of the whole system, as essentially they avoid the time and space conflicts between different trains, which is the decisive factor to the range of safety region. Therefore, a train rescheduling method is specially proposed in this part.

1) The principle and strategy of train rescheduling

When the railway system is in unbalanced state, strategies to restore the system need to follow certain principles.

a) Principles of train rescheduling

- Schedule the train in the original path and avoid detour and outage to the greatest extent;
- When detour is necessary, check the train and the line conform or not and choose the shortest one;
- Higher grade Trains can't be overtaken by lower ones;

- Passenger trains can't be overtaken by freight trains;
- The punctual trains have a higher priority.
- Passenger trains can arrival in advance but can't departure in advance.

b) Strategies for train rescheduling

- Detour, outage, reconnection and turn-back can be adopted when necessary;
- Change the section running time;
- Change the dwelling time in station;
- Change the overtaking station or time.

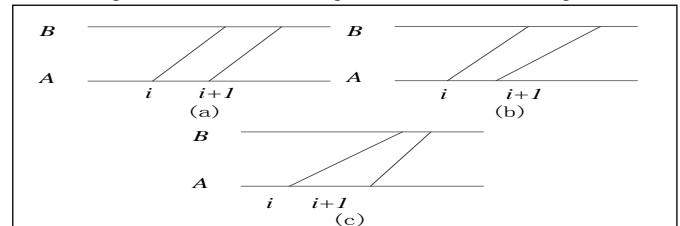
2) Rescheduling method

The process of train operation is discretized, so the rescheduling can be got one section by one section.

Paper [15] summarizes 3 rules for the events dispatching. A first-to-start dispatcher selects the next train to be moved based on the earliest start time. A first-to-finish dispatcher selects the next train to be moved based on the earliest finish time on its next segment. Other possible dispatchers can be created by setting the dispatch decision time for train i as $t_i = (1-\delta)u_i + \delta v_i$, where u_i is the start time for train i and v_i is its expected finish time on its next immediate segment and $\delta \in [0,1]$.

While the trains' grades are not considered in the dispatching rules mentioned before. As the grades are different between the neighbouring trains, there will be 3 situations: the neighbouring trains have the same grades (Fig.6(a)), higher grades train run after the lower grade train (Fig.6(b)), and lower grade train run after the higher grade train (Fig.6(c)).

Figure 6. Different tracking form of different train degree



When the actual start time of trains (AST) in each section is obtained, the timetable is got too. So the calculation of AST is the key of the problem. In this paper, AST is calculated by the formulas in Table III.

TABLE III. FORMULAS FOR THE ACTUAL START TIME

(a)	If $s_{i+1}'' - s_i \geq I$	then $s_{i+1} = s_{i+1}'', s_i = s_i$
	If $s_{i+1}'' - s_i < I$	Then $s_{i+1} = s_i + I, s_i = s_i$
(b)	If $s_{i+1}'' - s_i \geq I$	Then $s_{i+1} = s_{i+1}'', s_i = s_i$
	If $s_{i+1}'' - s_i < I$	Then $s_{i+1} = s_i + I, s_i = s_i$
(c)	If $s_{i+1}'' - s_i \geq I + t_i - t_{i+1}$	Then $s_{i+1} = s_{i+1}'', s_i = s_i$
	If $s_{i+1}'' - s_i < I + t_i - t_{i+1}$	Then $s_{i+1} = s_{i+1}'', s_i = s_{i+1} + I$

In Tab. III, s_i stands for AST, while s_i^* stands for the earliest start time (EST). The two concepts can be distinguished that AST is EST considering constrains between trains. EST can be got by two factors, a) the reckoning time according to AST and section running time in last section and the operation time in last station, b) the start time in the original timetable. We choose the bigger one as the result. It can be seen in (1).

$$s_k^* = \max(s_{k-1} + t_{k-1} + t_j, s_k^*) \quad (1)$$

Where, s_k^* stands for EST in section k, s_{k-1} stands for AST in section k-1, t_{k-1} stands for the running time in section k-1, and t_j stands for the operation time in station j.

On account of factors such as weather, track condition, equipment condition and etc., the velocity of trains is not constant. So we consider the pure running time as a variable number. The section running is depicted in (2).

$$t_p = \alpha_{p-1} \tau_q + \alpha_{p+1} \tau_t + t_p + \delta \quad (2)$$

Where, α is a 0-1 variable representing whether train stops in station or not, τ_q and τ_t stand for the addition time of start and stop, t_p stands for the pure running time, δ is a stochastic number.

The variation of section running time enriches the problem space, and we can find a better solution. The value of δ is vital to the quality of the result. R. Albrecht [16] made many experiment to obtain a more proper value in his doctoral dissertation, finding that when distributed normally (that is $\delta \in N(0, \alpha_T)$ and $\alpha_T = m/2$ [15], the result could be better. m stands for the section running time. The conclusion is still applied in this paper.

The algorithm is depicted in the following.

- Step1: Choose all the events in section I,
- Step2: Calculate the earliest start time of section i according to formula (1),
- Step3: Calculate the actual start time of the section event according to table III,
- Step4: Do $i=i+1$ until the last section,
- Step5: Repeat step 1 to step 5 N times (N is determined by decision maker, it can be 100 or another), thus we have N feasible schemes, and find the best solution according to the object function among the N feasible schemes,
- Step6: Draw the adjusted train diagram.

3) An experimental example of the method

The results of using the method before are discussed here for a representative example on Jin-qin passenger railway (approximately 260km with 9 stations), China. The case is based on real data and a scene with disorder is assumed.

The assumed scene: the section Junliangcheng north station to Binhai station suffered heavy rainfall during the period

13:00 to 17:00. And the allowed speed of the trains passed by then is 100 km/h. Because of the bad weather, 11 trains are late. So a quick adjustment of train timetable is needed.

We take the minimum deviation between the original timetable and the adjusted timetable as objective, then carry out the algorithm before with related data and the output objective distribution is shown in Fig.7. The results are normal distributed. The result with minimum deviation time is an ideal scheme and the rescheduled timetable is shown in Fig.8.

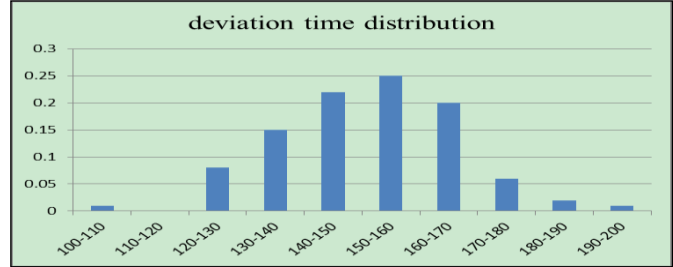


Figure 7. Objective distribution with variable running time

We can see that in Fig.9, D6795 is a train with lower grade compared to others and in order to cause larger deviation it is overtaken by train G1253 in Binhai station only. In this case the objective number is 109.3672 and the result can be got in an acceptable time. The method has also been applied with success to a range of test problems with various network sizes, number of trains and works well.

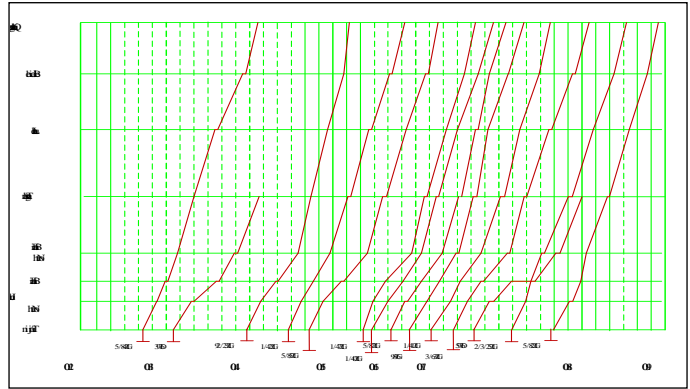


Figure 8. Train timetable with variable running time

IV. THE CORRESPONDING PREVENTION MEASURES

Besides the control measures, on the basis of the theory and analytical method of the P-SR accident model, we can further conclude the following preventive measures against accidents:

- (1) Strengthen the implement of technical engineering in the system changes and control measures parts. As the external disturbance is almost inevitable, to maintain system balanced state is the critical process to prevent an accident.
- (2) Strengthen the monitoring of the system running state and quantitative analysis of safety region, so as to timely reflect the safe state of the system. And then offer the safe state analysis and early warning information to provide basis for adopting corresponding control measures;
- (3) Take comprehensive and effective safety control measures based on safe state and early warning information,

and at the same time constantly monitor the system state to assess the effectiveness of safety measures to adjust inappropriate control measures in time.

(4) Strengthen the construction of emergency management and human emergency response. As human bears huge psychological pressure when the system works out of order after disturbance, they are likely to make inappropriate decisions or take unsuitable actions that may aggravate the reduction of system safety margin.

V. CONCLUSION

This paper presents a new accident model, perturbation-safety region accident-causing theory model, to analyze complex system, based on perturbation occurs theory and system theory. The model we proposed focuses particular attention on how to measure safe state of the system as a feedback to control measures and how the relative control measures are taken according to that feedback. Instead of analyzing safety in the context of preventing component failure, it addresses the continuous monitor and control task after perturbation. Accidents are seen as resulting from inadequate or inappropriate control measure during system design, daily operation and emergency response. The process of an accident is captured from (1) intrinsic nature of the dangerous resource in the system, and then (2) the perturbation brought up by the unsafe state and behavior that deviates system from safe space, to (3) the alarm and monitor part that uses safe margin to guide corresponding control measure with assessment of it. Ultimately, the accident can be understood in terms of why the control measure enforced in a disturbed system fails to stop the progress of it. Specifically in the model, the system safe state depict by safety region visualizes the course of an accident by safe margin, which evaluates how close the system is near to accident space. This allows controllers or decision makers to have the crucial feedback to adopt appropriate measures.

Hence, the P-SR model also overcomes the limitation that most accident models do not apply in real-time work. The results of the analysis not only contains static charts or figures, but also a dynamic system state diagram that monitors the system continuously, which could be implemented in real work to maintain safety. Through learning the progress of an accident, the notion of the model changes from the passive analysis after accident to the initiative safety restoration before accident. The necessity of this change lies in that the potential interactions between components in a system is rather complex that they are hard to understand and anticipate. So the common accident models, chain events or dynamic networks, focusing on how to prevent accident by exposing flaws in physic parts and behaviors with their interrelations, is not enough to keep up with the safety management needed in different kinds of system and perturbation. And yet the model we present builds a unified mathematical expression frame to describe the change brought up by multiple factors, which elevates the quantitative analysis ability for complex systems.

The validity of the model has been proved as the analysis and reconstruction of a typical railway accident in China case shows. To further illustrate the role that control measures take in a disturbed system, a railway emergency command method

is proposed in this paper to back up the safety restoration with respect to the perturbation in daily operation.

The concept of P-SR model is suitable to improve performance in safety management. But there are still problems to be solved before the application, like (1) the safety region of each different system should be specified; (2) the characteristic state variables are crucial to the whole analysis that omitted variable may also increase the risk of accidents; (3) massive amounts of data are needed to be collected and analyzed.

ACKNOWLEDGMENT

This study is sponsored by the 863 Program (2012AA112001) of China's Ministry of Science and Technology, and Specialized Research Fund for the Doctoral Program (20120009110035) of National Ministry of Education. The support of State Key Lab of Rail Traffic Control & Safety of Beijing Jiaotong University is also gratefully acknowledged.

REFERENCES

- [1] A. Kuhlmann, *An Introduction to Safety Science*, Germany, 1981.
- [2] Heinrich, H.W., Petersen, D., Roos, N., *Industrial accident prevention: a safety management approach*, fifth ed, The U.S.:Mcgraw-Hill,1980.
- [3] H. Xueqiu, *Safety Engineering*, China: China University of Mining and Technology,2000.
- [4] Nancy Leveson, "A New Accident Model for Engineering Safer Systems," *Safety science*, vol.42, No.4,pp.237-270, April,2004
- [5] China higher education committee in Safety Engineering Guidancs, *Security System Engineering*, China:China Coal Industry, 2002.
- [6] Paul M. Salmon, Natassia Goode, Frank Archer, Caroline Spencer, Dudley McArdle, Roderick J. McClure, "A System approach to examining disaster response: Using Accimap to describe the factors influencing bushfire response," *Safety science*, vol.70, .pp.114-122, December,2014.
- [7] YunXiao Fan, Zhi Li, JingJing Pei, Hongyu Li, Jiang Sun, "Applying system thinking approach to accident analysis in China: Case study of "7.23" Yong-Tia-Wen High-speed train accident," *Safety Science*, vol. 76, pp. 190-201, July 2015.
- [8] Rasmussen, J., "Risk management in a dynamic society: a modelling problem," *Safety Science*. Vol. 27, pp. 183-213 1997.
- [9] Hollnagel, E., *Investigation as an impediment to learning*. In: Hollnagel, E., Nemeth, C., Dekker, S. (Eds.), *Remaining Sensitive to the Possibility of Failure, Resilience Engineering Series*. The U.K.: Ashgate, Aldershot, 2008.
- [10] R Amalberti, "The paradoxes of almost totally safe transportation systems," *Safety Science*, vol. 37, pp. 109-126, March 2001.
- [11] Rogier Woltjer, Ella Pinska-Chauvin, Tom Laursen, Billy Josefsson, "Towards understanding work-as-done in air traffic management safety assessment and design," *Reliability Engineering & System Safety*, March 2015. [Online]. Available: Elsevier, <http://www.sciencedirect.com>.
- [12] X. Ancheng, Wu, F., F., L. Qiang, M. Shengwei, "Power system dynamic security region and its approximations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol.53, pp.2849-2859, December 2006.
- [13] Q. Yong, S. Jingxuan, Z. Yuan, Z. Shengzhi, J. Limin, "Online Security Assessment of Rail Vehicles in Service Status based on Safety Region Estimation," *Journal of Central South University (Science and Technology)*, vol.44, pp .195-200, July 2013.
- [14] The State Council of China, *Yongwen Railway Line Major Traffic Accident Investigation Report*, China, December 2011.
- [15] Albrecht A R, Panton D M, Lee D H., "Rescheduling rail networks with maintenance disruptions using Problem Space Search," *Computers & Operations Research*, vol. 40, pp. 703-712, September 2013.
- [16] Amie R. Albrecht, "Integrating railway track maintenance and train timetables", University of South Australia, 2009.

Statically-Guided Fork-based Symbolic Execution for Vulnerability Detection

Yue Wang, Hao Sun, Qingkai Zeng

State Key Lab for Novel Software Technology, Nanjing University
Department of Computer Science and Technology, Nanjing University
Nanjing 210023, China
wxywang89@163.com, shqking@gmail.com, zqk@nju.edu.cn

Abstract—Fork-based symbolic execution would waste large amounts of computing time and resource on *invulnerable paths* when applied to *vulnerability detection*. In this paper, we propose a statically-guided fork-based symbolic execution technique for vulnerability detection to mitigate this problem. In static analysis, we collect all *valid jumps* along *vulnerable paths*, and define the priority for each program branch based on the ratio of vulnerable paths over total paths in its subsequent program. In fork-based symbolic execution, path exploration can be restricted to vulnerable paths, and code segments with higher proportion of vulnerable paths can be analyzed in advance by utilizing the result of static analysis. We implement a prototype named *SAF-SE* and evaluate it with ten benchmarks from GNU Coreutils version 6.11. Experimental results show that *SAF-SE* outperforms KLEE in the efficiency and accuracy of vulnerability detection.

Keywords—fork-based symbolic execution; static analysis; vulnerability detection; program analysis

I. INTRODUCTION

Symbolic execution was first proposed by James C. King [1] in 1976. Fork-based symbolic execution uses symbolic values as inputs to execute target programs and replaces concrete program operations with ones that manipulate symbolic values during the execution. When program execution branches based on symbolic values, it follows each valid branch and collects the branch condition as the constraint of the corresponding path. When one path terminates or hits a bug, a test case will be generated by solving the collected constraints. Symbolic execution has two advantages: 1) having high code coverage and 2) producing no false positives.

Recently, fork-based symbolic execution has been applied to the field of *vulnerability detection*. The key challenge lies in that the goal of vulnerability detection is to expose vulnerable code as soon as possible and *vulnerable paths*, i.e. paths involving vulnerable code, only occupy a small proportion in programs, while fork-based symbolic execution selects branch blindly, leading to a considerable waste of computing time and resource on exploring *invulnerable paths*. Besides, the vulnerability detection accuracy would also be affected, i.e. generating false negatives, if computing time and resource is limited in real-world scenarios. Furthermore, *path explosion* would worsen both the efficiency and accuracy of vulnerability detection if target programs are in large scale.

To address this issue, we propose and implement a statically-guided fork-based symbolic execution technique for vulnerability detection, named *SAF-SE*. Static analysis process marks vulnerable paths and collects all valid jumps along them. These valid jumps would restrict symbolic execution to vulnerable paths only, and generate test cases which can violate the security constraints of sensitive operations. Furthermore, we define the priority for each program branch based on the ratio of vulnerable paths over total paths in its subsequent program. These branch scores are used by execution state selector to determine the priority of each execution state. Therefore, program segments with higher proportion of vulnerable paths will be explored first and more vulnerable code will be detected in the circumstance of limited computing time or resource. Hence, *SAF-SE* can not only accelerate fork-based symbolic execution process but also improve the accuracy of vulnerability detection.

This paper makes three contributions. First, we propose a statically-guided fork-based symbolic execution technique for vulnerability detection, in which we restrict fork-based symbolic execution on vulnerable paths. Second, we score program branches based on the ratio of vulnerable paths in subsequent program. Hence, code segments with higher proportion of vulnerable paths would be analyzed earlier. Third, we implement a prototype name *SAF-SE* and evaluate it with 10 benchmarks from GNU Coreutils 6.11. Experimental results show *SAF-SE* can improve vulnerability detection efficiency, and reduce false negatives when time and resource is limited.

II. DESIGN OF SAF-SE

Figure 1 illustrates the architecture of *SAF-SE*. It consists of three components: *graph generation* module, *static analysis* module and *fork-based symbolic execution* module. Note that users can define sensitive operations and corresponding security constraints in *user-defined configuration file*.

A. Graph Generation Module

Graph generation module reads LLVM bytecode file as input and generates the call graph and control flow graphs (CFGs). The call graph and CFG generation process in LLVM Utils doesn't consider dynamic link library functions. Therefore, we utilize a *light-weight symbolic executor* to obtain a relatively complete program. In it, we simulate the link process by executing the target program symbolically with the simplest

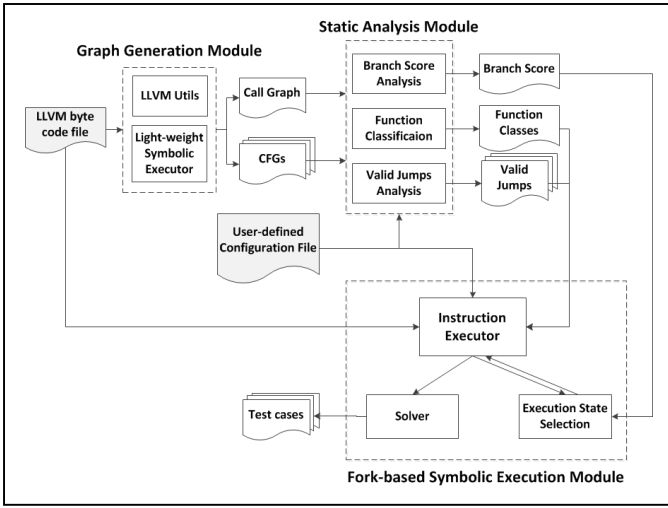


Figure 1. Architecture of SAF-SE

symbolic strategy, i.e. one symbolic input sized of one character in order to get dynamic library functions in records. Considering the small time consumption, i.e. less than 10 seconds, we call it *light-weight symbolic executor*.

B. Static Analysis Module

Static analysis module at first locates sensitive operations in target programs and divides functions into three categories. Then *valid jumps analysis* collects all valid jumps by marking the conditional values of each branch instruction that can lead to sensitive operations. At last, *branch score analysis* calculates the score of program branch according to the ratio of vulnerable paths over total paths in its subsequent program.

1) Function Classification

We define two attributes for each function, i.e. *vul-related* and *vul-lead*.

Definition 1. Function f is *vul-related*, if there are sensitive operations in f , or f calls another *vul-related* function.

Definition 2. If function h invokes function f at call site loc , and there are sensitive operations or *vul-related* function calls on the paths from loc to the exits of function h , then function f is *vul-lead*.

To calculate *vul-related* attribute, initially, we mark functions with sensitive operations inside as *vul-related*. Then, callers of *vul-related* functions are also marked as *vul-related*. To calculate *vul-related* attributes, first we locate the positions of sensitive operations and *vul-related* function call sites as *locs* in each function. Then, we initialize all the functions called between the function entry and *locs* as *vul-lead*. At last, all callees of *vul-lead* functions are also marked as *vul-lead*.

According to the attributes, functions in target programs can be divided into three categories, i.e. $T1$ ~ $T3$, and different execution strategies would be applied to different categories.

a) $T1$: *vul-lead* = true: sensitive operations would be invoked after $T1$ functions. Since operations within $T1$ function might affect the sensitive operations afterward, all paths inside would be executed symbolically to gain conservative results.

Algorithm 1

input: $CFG_{fn}, V_{target-op}$

output: $Set_{valid-jmp}$

procedure calValidJmps ($CFG_{fn}, V_{target-ops}$)

```

1   $V_{sen-BBs} = collectSensitiveBBs(CFG_{fn}, V_{target-op});$ 
2  for each BasicBlock  $bb \in V_{sen-BBs}$  do
3     $V_{parent-BBs} = collectParentBBs(CFG_{fn}, bb);$ 
4    for each BasicBlock  $pbb$  in  $V_{parent-BBs}$  do
5      Instruction  $inst = last\ branch\ instruction\ in\ pbb;$ 
6       $validChoice = getValidCondition(pbb, bb);$ 
7       $Set_{valid-jmp}.insert(\langle fn, inst, validChoice \rangle);$ 
8       $V_{sen-BBs}.insert(pbb);$ 
9  return  $Set_{valid-jmp};$ 

```

b) $T2$: *vul-related* = true and *vul-lead* = false: $T2$ functions have sensitive operations inside and have no sensitive operation after the execution of them. Hence, invulnerable paths inside can be pruned in symbolic execution.

c) $T3$: *vul-related* = false and *vul-lead* = false: $T3$ functions neither involve sensitive operations, nor have sensitive operations afterward. Hence, symbolic executor would terminate the execution process for $T3$ function calls.

2) Valid Jumps Analysis

Valid jumps analysis aims to collect the *valid jumps* of each branch instruction in $T2$ functions. *Valid jump* is the conditional value of branch instruction which can lead execution to sensitive operations. Algorithm 1 illustrates this process for function fn . $V_{target-op}$ refers to the set of sensitive operations and *vul-related* function calls. Tuple $\langle f, inst, choice \rangle$ is used to denote one valid jump, i.e. the instruction $inst$ in function f would lead to sensitive operations under the conditional value $choice$. $Set_{valid-jmp}$ stores all the collected valid jumps.

Our valid jumps analysis is at basic block granularity. Initially, basic blocks involving $V_{target-op}$ are marked as sensitive (line 1). Then, for each sensitive basic block bb , we fetch each of its preceding basic block pbb (line 3) and set the branch from pbb to bb as a valid jump (lines 5 to 7). At last, pbb is also marked as sensitive (line 8). This process will continue until all sensitive basic blocks have been analyzed.

3) Branch Score Analysis

In branch score analysis, for each program branch, first we count the number of total paths and vulnerable paths, then score the branch based on the ratio of vulnerable paths over total paths in its subsequent program. These scores would be further used by execution state selector to explore the branch with higher proportion of vulnerable paths in advance. Note that we count the loop as one path when calculating the number of paths due to the lack of actual execution times of the loop structure in static analysis.

C. Fork-based Symbolic Execution Module

Fork-based symbolic execution module explores vulnerable paths following the branch scores and generates test cases which can violate security constraints for sensitive operations.

Generally speaking, it consists of three main parts: *instruction executor*, *execution state selector* and *constraint solver*.

1) *Instruction Executor*

We modify the instruction executor to analyze vulnerable paths with the results of function classification and valid jumps analysis. When we deal with sensitive operations, a verify process *check()* will be used to check if current constraints violate security constraints. If so, a test case would be generated by the constraint solver and reported to users.

When executing *Call* instructions, we check whether the callee is T3 function. If so, we would terminate current execution process and remove the execution state from the execution state pool based on the analysis in Section II-B-1).

As for *Branch* instructions and *Switch* instructions, we at first check whether current function belongs to T1 function. If so, we follow the original fork-based symbolic execution process. If current function is T2 function, execution flow can only be transferred to the valid succeeding basic blocks according to the result of valid jumps analysis. For each valid branch, we construct a new execution state by copying the current execution state, changing instruction pointer *pc* to the valid destination instruction, and adding condition expression into constraint set. At last, we insert the new execution states into the execution state pool.

2) *Execution State Selector*

Execution state selector aims to select an execution state from the execution state pool. Since existing selection strategies, e.g. depth-first search (DFS), breadth-first search (BFS), and covering new focus on program coverage, they cannot accelerate the process of vulnerability detection. Hence, we design a new selection strategy for vulnerability detection. Leveraging the results of branch score analysis, we select the execution state in the order of scores. In this way, code segments with high proportion of vulnerable paths would get analyzed in advance, accelerating vulnerable paths exploration and explore as many vulnerable paths as possible with limited computing time and resource.

III. IMPLEMENTATION AND EVALUATION

A. *Implementation Details*

We have implemented a prototype named *SAF-SE*. In it, we use a fork-based symbolic executor with one symbolic argument, whose size is one character, as the light-weight symbolic executor, and LLVM-3.1 utils to generate call graph and CFGs. As for the static analysis part, we implement a LLVM optimization pass written in about 1,600 lines of C++ on call graph and CFGs. In fork-based symbolic execution module, we adopt KLEE [2] and modify its instruction executor and the execution state selector based on previous discussion.

B. *Experimental Setup*

To evaluate the effectiveness of *SAF-SE*, we applied it on ten programs from GNU Coreutils version 6.11, and compared the results with KLEE [2]. In our experiments, we set seven library function calls as sensitive operations, including *alloc*, *malloc*, *realloc*, *calloc*, *memcpy*, *memccpy* and *memset*. All experiments were run on a machine with 3.20GHz Intel(R)

Core(TM) i5-3470 processor and 4G of memory, running 64-bit Linux 3.2.0.

C. *Results of Static Analysis*

Table 1 shows the experimental results of static analysis on the ten benchmarks. The time cost (Column 2) in static analysis is negligible, averaging about 0.389s. Columns 3 to 5 show the distributions of three types of functions. We can observe that T2 functions, in which invulnerable paths can be pruned, account for the largest proportion, about 48.4% on average. Column 6 indicates the number of the basic blocks that can be pruned by static analysis, including all the basic blocks in T3 functions and those along invulnerable paths in T2 functions. On average, 21.8% of all basic blocks are free of symbolic execution.

D. *Results of SAF-SE*

To assess the effectiveness of *SAF-SE*, we look into the following two aspects: 1) we applied *SAF-SE* and KLEE on five benchmarks with the same arguments, respectively. For each benchmark, both *SAF-SE* and KLEE completed the whole symbolic execution process, and we assessed the reduction in execution time and executed instructions of *SAF-SE* over KLEE; 2) we applied *SAF-SE* and KLEE on the other five benchmarks with the same arguments, and we limited the execution time to 60 minutes, so as to assess the sensitive operation coverage promotion of *SAF-SE* over KLEE.

Table II shows the experimental results of the first aspect. Columns 2 to 4 show the results of KLEE, including the execution time, the number of analyzed instructions and the number of analyzed sensitive operations, and Columns 5 to 7 show those of *SAF-SE*. On average, *SAF-SE* achieved about 23.52% execution time reduction and about 23.17% analyzed instruction reduction over KLEE. In a word, *SAF-SE* can spend less execution time and execute fewer instructions than KLEE in completing the symbolic execution process without missing any sensitive operations.

Table III describes the experimental results of the second aspect. The meanings of the columns are similar to those in Table II. Each benchmark was run for 60 minutes with the same symbolic arguments: `--sym-args 1 5 10 --sym-files 2 100`, which means the number of symbolic arguments are from 1 to 5, and the length of each symbolic arguments is up to 10 characters. Meanwhile, we use two symbolic files which are not longer than 100 characters. From the results we can see that, *SAF-SE* executed 1.70x of instructions that KLEE executed, and discovered 1.37x of sensitive operations that KLEE covered in the same execution time. It is worth noting that the effectiveness in sensitive operation coverage promotion is highly dependent on the structure of each program and on the distribution of sensitive operations. We can conclude that *SAF-SE* can explore more vulnerable paths under limited execution time than KLEE.

The Experimental result proves *SAF-SE* can improve vulnerability detection efficiency of fork-based symbolic execution by pruning invulnerable paths in advance. Moreover, *SAF-SE* can reduce false negatives in the circumstance of limited computing time and resource with the help of branch scores from static analysis.

TABLE I. RESULTS OF STATIC ANALYSIS

program	time(s)	T1 (num/rate)	T2 (num/rate)	T3 (num/rate)	Prune BBs (num/rate)
mkdir	0.410	167/0.498	154/0.460	14/0.042	750/0.209
mkfifo	0.391	143/0.451	160/0.505	14/0.044	796/0.236
mknod	0.379	148/0.460	160/0.497	14/0.043	845/0.242
paste	0.363	146/0.458	159/0.498	14/0.044	777/0.226
ptx	0.685	196/0.513	167/0.437	19/0.050	886/0.158
seq	0.437	148/0.454	163/0.500	15/0.046	810/0.234
chmod	0.354	210/0.532	161/0.409	23/0.058	851/0.198
echo	0.281	132/0.423	165/0.529	15/0.048	841/0.251
basename	0.287	142/0.444	163/0.509	15/0.047	790/0.237
cat	0.300	148/0.460	159/0.494	15/0.046	816/0.234
AVG	0.389	158.0/0.470	161.1/0.484	15.8/0.046	816.2/0.218

TABLE II. RESULTS OF BENCHMARKS COMPLETED SYMBOLIC EXECUTION

program	KLEE			SAF-SE		
	time(s)	instruction	sensitive op	time(s)	instruction	sensitive op
echo	1548.01	40878183	20409	1045.72(-32.45%)	29702739(-27.34%)	20409
chmod	315.48	63130620	28425	243.52(-22.81%)	48300940(-23.49%)	28425
mkfifo	323.71	61459653	26275	228.82(-29.31%)	48493576(-21.10%)	26275
mknod	320.3	54853060	21937	259.67(-18.93%)	41718084(-23.95%)	21937
basename	25.45	5264958	2337	21.85(-14.16%)	4213381(-19.97%)	2337

TABLE III. RESULTS OF BENCHMARKS RUN FOR 60 MINUTES

program	KLEE			SAF-SE		
	time(s)	instruction	sensitive op	time(s)	instruction	sensitive op
paste	3832.64	47913926	29752	3644.73	163029964(+240.26%)	61813(+107.76)
ptx	3706.1	99826914	40368	3718.47	102729476(+2.91%)	41703(+3.31%)
cat	4169.63	11818118	16565	4179.68	11868664(+0.43%)	16921(+2.15%)
seq	3669.66	81698491	35033	3668.82	103876566(+27.15%)	40830(+16.55%)
mkdir	3699.58	315383144	50299	3697.19	574760703(+82.24%)	76906(+52.90%)

IV. RELATED WORK

KLEE [2] is a widely used fork-based symbolic execution tool evolves from EXE [3]. Vitaly Chipounov et al. [4] proposed selective symbolic execution and implemented S2E by adopting KLEE as symbolic executor and using QEMU [5] to simulate execution environment. Combining symbolic execution with concrete execution, Patrice Godefroid et al. proposed the first concolic symbolic execution tool SAGE [6] for binary code. Symbolic execution has been widely used in vulnerability detection. SmartFuzz [7] leverages concolic symbolic execution to find integer bugs in x86 binary programs. Crashmaker [8] optimized the generational search algorithm in SAGE. However, these techniques still have to traversal the whole program even when detecting specific sensitive operations, while *SAF-SE* can improve the efficiency and accuracy of vulnerability detection by restricting path exploration on vulnerable paths.

V. CONCLUSION

In this paper, we propose a statically-guided fork-based symbolic execution technique for vulnerability detection and developed a prototype *SAF-SE* to restrict path exploration on vulnerable paths and to explore code segments with higher proportion of vulnerable paths earlier by utilizing the results of static analysis. We evaluated *SAF-SE* with ten benchmarks from GNU Coreutils version 6.11, and compared it with KLEE. The experimental results show that, *SAF-SE* improves the efficiency of vulnerability detection a lot, and reduces

generating false negatives in the circumstances of limited computing time and resource.

REFERENCES

- [1] J.C.King, "Symbolic execution and program testing", ;inproceedings of Communications of the ACM, 1976, pp.385-394
- [2] C.Cadar, D.Dunbar, and D.Engler, "KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs", ;inproceedings of OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation, 2008, pp.209-224
- [3] C.Cadar, V.Ganesh, P.M..Pawlowski, D.L..Dill, and D.R..Engler, "EXE: automatically generating inputs of death", ;conference of Computer and Communications Security, 2006, pp.322-335
- [4] V.Chipounov, V.Kuznetsov, and G.Candea, "S2E: a platform for in-vivo multi-path analysis of software systems", ;inproceedings of Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, 2011, pp.265-278
- [5] F.Bellard, "QEMU, a fast and portable dynamic translator", ;inproceedings of ATEC '05 Proceedings of the annual conference on USENIX Annual Technical Conference, 2005, pp.41-41
- [6] P.Godefroid, M.Y..Levin, and D.A..Molnar, "Automated Whitebox Fuzz Testing", ;conference of Network and Distributed System Security Symposium, 2008, pp.-1—1
- [7] D.Molnar, X.Cong.Li, and D.A..Wagner, "Dynamic test generation to find integer bugs in x86 binary linux programs", ;inproceedings of SSYM'09 Proceedings of the 18th conference on USENIX security symposium, 2009, pp.67-82
- [8] Bing Chen, Qingkai Zeng, and Weiguang Wang. "Crashmaker: an improved binary concolic testing tool for vulnerability detection." inproceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, 2014, pp.1257-1263

How Does Defect Removal Activity of Developer Vary with Development Experience?

Reou Ando, Seiji Sato, Chihiro Uchida,
Hironori Washizaki, and Yoshiaki Fukazawa
Department of Computer Science and
Engineering
Waseda University
Tokyo, Japan
Email: waseda-reou@suou.waseda.jp,
r0d8h8i0h@asagi.waseda.jp,
c.u.0224@ruri.waseda.jp, {washizaki,
fukazawa}@waseda.jp

Sakae Inoue, Hiroyuki Ono, Yoshiiku Hanai,
Masanobu Kanazawa, Kazutaka Sone,
Katsushi Namba, and Mikihiko Yamamoto
Fujitsu Limited
Kanagawa, Japan
Email: {inoue.sakae, ono.hiro, hanai.yoshiiku,
kanazawa.masano, sone.kazutaka, nanba,
yamamoto.mikihi}@jp.fujitsu.com

Abstract—Because developers significantly impact software development projects, many researchers have studied developers as a means to improve the quality of software. However, most works have examined developers in a single project, and research involving multiple projects has yet to be published. Herein we propose an analysis method which investigates whether an evaluation of developers based on individual experience is feasible when targeting more than one project by the same organization transversely. Our method deals with the logs of the version control system and the bug tracking system. To support this method, we also propose two models to evaluate developer, the defect removal overhead rate (DROR) and developer’s experience point (EXP). The results reveal the following. 1) DROR cannot be used to compare different projects in the same organization. 2) There is certainly a difference in DROR’s between experienced and inexperienced developers. 3) EXP should be a useful model to evaluate developers as the number of projects increases. The data obtained from our method should propose the personnel distribution measures within the development framework for future developments, which might lead to improve the quality of software.

I. INTRODUCTION

In software development projects, developers and organizations are said to significantly impact software [1-15, 17]. A 1968 study on the organization when analyzing software quality resulted in Conway’s law [2], which states that “organizations that design systems are constrained to produce systems which are copies of the communication structures of these organizations.” Recently, researchers have examined defect prediction in software using metrics based on hypotheses formed by the structure of an organization [12], and have investigated the effects of software in a project involving multiple organizations due to mergers and acquisitions [14], etc. Such studies have found that organizational structures greatly influence software quality [2, 3, 12, 14].

On the other hand, research on developers has proposed techniques to improve the prediction of potential defects in software by utilizing the quality of the developer. The quality of the developer is defined as how much his commits lead to defects in a project [17]. Defect prediction using metrics, such as the number of commits and LOC for each developer [6], assesses the impact of

developers on the quality and reliability of software [1, 4-9, 11, 13, 15, 17].

Most studies focus on the organizational structure and the quality of the developers with respect to a single project or a group project involving different organizations. However, the results across multiple projects by the same organization have yet to be published. With regard to the experience of developers who belong to the same organization, it is easy to imagine that the development experience in past projects affects later software development. In fact, although the target of their research was a single project, A. Mockus et al. [11] found that developer’s experience significantly affects the possibility of defects; more experienced developers tend to have fewer defects.

If the results about developers based on past development experience are obtained by traversing multiple projects, it may be possible to improve a new project by structuring it so that is similar to developers’ previous experiences. Moreover, assuming a developer with little development experience introduces more defects, the development system should be arranged so that inexperienced developers work with experienced developers. This should improve the quality of software while simultaneously educating inexperienced developers. Therefore, we propose a technique to evaluate developers by analyzing their previous experiences from logs stored in the version control system and the bug tracking system in multiple projects. To determine how the defect removal activity of developers varies with development experience, we divided the issue into evaluable components. Hence, we formulated our study in the form of three research questions:

- *RQ1: As an organization gains project experience, does the defect removal overhead rate (DROR) of developers tend to decrease?*
- *RQ2: Is there difference in DROR based on development experience?*
- *RQ3: Is there difference in DROR between developers based on experience in a similar project?*

In order to respond to these research questions, we carried out evaluation experiments using our method. The

subjects of our study are developers in a real company involved in three projects, which do not overlap in the development periods.

The contributions of this study are:

- A method to evaluate developers based on past development experience using logs stored in the version control system and the bug tracking system.
- Understanding the trend of DROR based on developer experience.
- Obtaining resources to help improve measures of personnel distribution within the development framework for future developments.

The rest of the paper is organized as follows. Section 2 presents the background of our study through related work. Section 3 introduces our analysis method to address the problem described in Section 2. Then two analysis models to support our method are proposed in Section 4. In Section 5, we conduct experiments to evaluate our method and investigate the proposed research questions. Next Section 6 explains summary of findings and the practical application of our method. Finally we describe the conclusion in Section 7.

II. BACKGROUND AND RELATED WORK

A. Prior works focusing on developers

In software quality analysis, several works propose methods to predict defects in software based on the characteristic of developers [1, 4-9, 11, 13, 15, 17]. For example, Kamei et al. [6] observed the histories of developers commits. They proposed change measures, which extract the number of modified files recorded for each commit, lines of code added, and whether or not the change is a defect fix, etc. They found that by predicting software defects through change measures, high-risk fixes and the cost of high-quality software could be reduced.

Matsumoto et al. [9] extracted metrics such as the number of commits and LOC for each developer from the logs of the version control system. They supposed that these are useful for fault-prone analysis, which specifies the module containing defects. Besides, Y. Wu et al. [16] defined the quality for each developer from the proportion of commits that introduce defects into a project. They found that using their proposed eight metrics as parameters as lead to better fault-prone analysis compared to traditional process metrics.

B. One of the problems in related works

Developer experience varies by the individual. Numerous works deal with it [5-8, 11, 15], but the research focuses on evaluating a single project or a group of different organizations. Research on multiple projects in the same organization has yet to be published. Most prior works probably evaluate a single project, even though they considered developer experience.

If developers with experience are compared to those without experience, it is conceivable that there will be differences. In addition, it is possible that the type of experience leads to differences among experienced developers. Therefore, the research aims to evaluate

developers based on their development experience in multiple projects within the same organization in a cross-sectional way.

III. ANALYSIS METHOD

The participants in our study are developers involved in large-scale projects in an organization that uses a version control system and a bug tracking system.

Our analysis involves the following steps:

- (i) Extract logs from the version control system and the bug tracking system used in completed projects.
- (ii) Collect the names of developers, the number of files they changed, the names of the absolute path that they changed files, and the number of changes in them from the log of version control system. In addition, identify files recorded as defect fixes after detecting the defect; that is, files related with a defect (hereinafter referred to as *defect files*), from the logs of the version control system and the bug tracking system. Then collect the name of developer who changed defect files and the number of changed defect files.
- (iii) Gather the number of changed files, the changed absolute path's name and its number, and the number of changed defect files by developer name.
- (iv) Calculate each developer's *defect removal overhead rate* (DROR), which is detailed in Section 4, from the number of changed files for each developer.
- (v) Repeat steps (i) to (iv) for each completed project.
- (vi) If a developer's name exists in different projects, consider the developer to be experienced in later projects. Then calculate the *developer's experience point* (EXP) in the project, which is detailed in Section 4.

Finally, analyze each developer based on the gathered data. Incidentally, we assumed that a function should be implemented not by single file, but by all files included in the absolute path, which is why we use the number of changed absolute paths and not the number of changed files. In this method, the number of changed files includes the number of changing defect files.

IV. ANALYSIS MODEL

It seems important to prepare an indicator to link developers with the number of defect to evaluate developers individually. In this paper, we present a metric called *defect removal overhead rate* (DROR) for each developer. Moreover, in order to examine precisely what area and how much ability a developer has acquired, we also suggest a measure named *developer's experience point* (EXP) for each experienced developer.

A. Defect removal overhead rate (DROR)

In a large-scale development, it is important that people who are not engaged in implementation are involved in detecting defects. For this reason, it is probable that developers differ from testers. A developer who modifies certain defect files should have changed it because he induced the defects that testers requested to be fixed.

Table 1. Example each d_{p_1} 's DROR calculation

Date	d_{p_1}			P1			
	x	y	z	f ₁	f ₂	f ₃	f ₄
2015/1/10	✓			○	○	○	
2015/1/11		✓				⊙	
2015/1/12			✓				⊙
2015/1/15		✓		●	●	◦	
2015/1/16			✓				◦
$DROR(d_{p_1})$	$\frac{0}{3}$	$\frac{3}{4}$	$\frac{1}{2}$	○: Changed file ⊙: Changed file (defect occurred) ●: Defect-fixed file ◦: Defect-fixed file (defect removed)			

Hence, we assume that the person who injected defects into a file is the person who changed it. This assumption is used to define defect removal overhead rate (hereinafter referred to as DROR) of a developer.

DROR of a developer is calculated as the proportion of fixing defect files compared to the total number of files that he changed. When developer d_R involved in project R , d_R 's DROR is defined as

$$DROR(d_R) = \frac{NDF_{d_R}}{NF_{d_R}} \quad (1)$$

- R : The project which evaluates developer
- d_R : Developer who involved in R
- NF_{d_R} : The total number of d_R changing file in R
- NDF_{d_R} : The number of d_R fixing defect file in R

Equation (1) can also be understood as the probability that the developer fixes a defect file when changing a file. The higher DROR, the more the developer is evaluated badly. It is because developers who write low-quality code should change more files related with a defect than developers who write high-quality program. Table 1 gives an example of each d_{p_1} 's DROR calculation. When developer x changed files f_1 , f_2 and f_3 on 2015/1/10, DROR of x is calculated as $\frac{0}{3}$ because he didn't fix defect files. Besides, developer y changed file f_3 , in which he might have induced a defect at that time, on 2015/1/11. And he fixed defect files f_1 , f_2 and f_3 on 2015/1/15. Then, DROR of y is measured as $\frac{3}{4}$. In addition, developer z changed file f_4 , in which he might have introduced a defect, on 2015/1/12. If he fixed file f_4 on 2015/1/16, DROR of z is figured out as $\frac{1}{2}$. If comparing these developers, developer y is evaluated the worst.

B. Developer's experience point (EXP)

Developer's experience point (hereinafter referred to as EXP) is measurement that considers his development experience. When there is developer $d_{P,R}$ who has experienced past projects P and is involved in the project R , $d_{P,R}$'s EXP in R is defined as

$$EXP(d_{P,R}) = \sum_{p_i \in R} C_R(p_i) \times C_P(p_i) \quad (2)$$

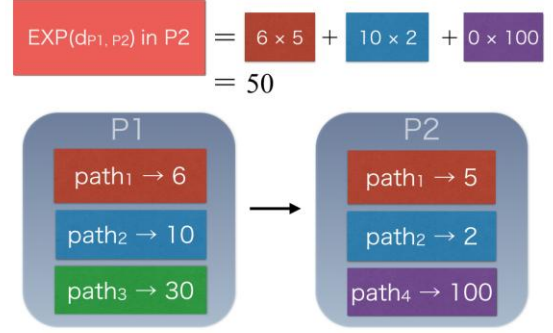


Figure 1. Example d_{p_1,p_2} 's EXP calculation

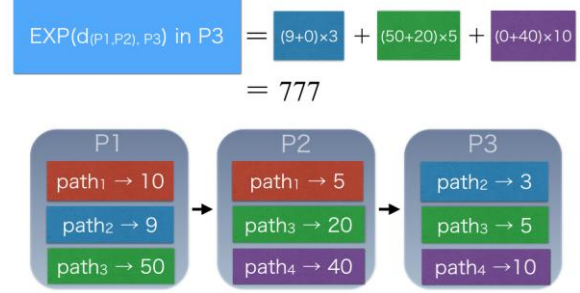


Figure 2. Example $d_{(p_1,p_2),p_3}$'s EXP calculation

- P : Past projects
- R : The project which evaluates developer
- $d_{P,R}$: Developer who experienced P and R
- $C_R(p_i)$: The number of appearing absolute path p_i which $d_{P,R}$ changed in R

Equation (2) means that if absolute path p_i in which $d_{P,R}$ changed files in R also exists in P , the number of changing files in p_i in R , defined $C_R(p_i)$, is weighted by that in P , which defined $C_P(p_i)$. The higher EXP, the more experience the developer has. The thought of (2) is developed by referring to A. Mockus et al. [11] and Y. Kamei et al. [6]. Figure 1 gives an example calculation of EXP when d_{p_1,p_2} changed the contents of path₁ 6 times, path₂ 10 times, and path₃ 30 times in P1. Moreover, he also edited path₁ 5 times, path₂ 2 times, and path₄ 100 times in P2. Then, his EXP in P2 is calculated as 50 (i.e. $(6 \times 5) + (10 \times 2) + (0 \times 100)$). Note that path₃ in P1 is not used in this example because he did not change it in P2. Figure 3 gives another example of EXP calculation. When $d_{(p_1,p_2),p_3}$ changed the contents of files shown in Fig. 2, his EXP in P3 is figured out as 777 (i.e. $((9+0) \times 3) + ((50+20) \times 5) + ((0+40) \times 10)$).

There are two purposes to define EXP using Eq. (2). First developers can be separated according to development experience. Although many developers have some experience, the amount likely varies by developer. If they are treated equally, the evaluation of developers can be mistaken. The other purpose is to consider developers with some experience but not in the type of project. As a result of taking these purposes into account, we adopted the system that $C_R(p_i)$ is weighted by $C_P(p_i)$. Scale type of EXP is ratio scale; EXP takes value from 0 indicating that the corresponding developer has no experience.

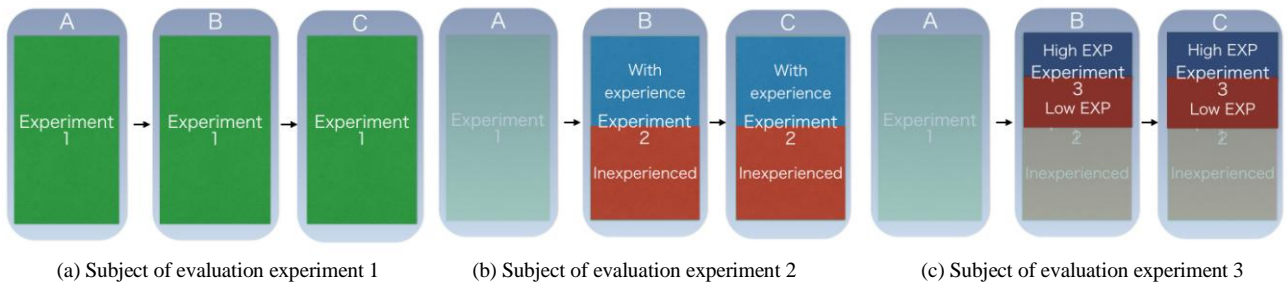


Figure 3. Subject of each evaluation experiment

V. EMPIRICAL EVALUATION

To evaluate the proposed method in this paper, we analyzed hundreds of developers who were involved in three different completed projects of embedded system development in a real company. In these three projects, developers released software that is enhanced seasonally. Thus, it is reasonable that the order of time is about the same for each project and it is unlikely that the projects were carried out simultaneously. In the following, three projects by this company are named project A, B, and C in order of time. Incidentally, the scale of this company's project ranges from 200,000 LOC to 300,000 LOC per project. There were hundreds or a few thousands of defects and thousands of commits per project¹.

A. Experiment

We obtained the logs of the version control system (Perforce²) and the bug tracking system (Prismy³) used in projects A, B, and C. Then, we gathered data for developers in each project according to procedure described in Section 3. Next, we set up evaluation experiments to correspond to each research question presented in Section 1. Finally, we divided the developers into several groups (Fig. 3).

- **Evaluation experiment 1 corresponding to RQ1** divides the developers into three groups depending on whether they are involved in project A, B, or C.
- **Evaluation experiment 2 corresponding to RQ2** divides the developers into two groups according to whether they have experience in previous projects.
- **Evaluation experiment 3 corresponding to RQ3** divides developers into two groups with median of EXP as a boundary on those who have experience in projects B and C.

With respect to the results, we created boxplots and graphs of the empirical cumulative distribution function, that is to say ECDF, for DROR of a developer for each evaluation experiment. Reading the vertical axis of an ECDF graph when the horizontal axis is fixed allows the proportion of developers who have DROR up to a value that the horizontal axis indicates to be determined. On the other hand, if the graph is read through the horizontal axis with the vertical axis fixed, the maximum DROR can be grasped in proportion of developers.

¹ Due to a confidentiality agreement, we do not show precise numbers.

² <http://www.perforce.com>

³ <http://www.tjsys.co.jp/page.jsp?id=742>

B. Results and discussion

RQ1: As an organization gains project experience, does the DROR of developers tend to decrease?

Figure 4 shows a boxplot and an ECDF of DROR in evaluation experiment 1. Many developers in project A have a higher DROR than those in project B and C. Considering this and the fact that projects B and C are derived from project A, DROR seems to depend on organization experience. However, comparing ECDF of project B with that of project C shows that the proportion of developer in project B with a 0.2 or less DRORs is more than that in project C. Furthermore, comparing the boxplot of project B and that of project C indicates that the DROR of project C is more scattered than that of project B, suggesting that the DROR of developers varies with factors other than their development experience. It might be because some of developers are already at their peak and did not improve significantly.

These findings show that we cannot affirm that DROR tends to decrease as an organization experiences projects.

RQ2: Is there difference in DROR based on development experience?

Figure 5 shows the boxplots of the DROR in evaluation experiment 2. There is a gap in the DROR's for both project B and C according to developer experience. In addition, the width of boxplots for DROR of inexperienced developers in project B differs from that in project C, but the width of the boxplots of experienced developers in project B is in good agreement with that in project C. These results suggest that experienced developers are free not influenced by changes in the development system or software design in between projects B and C.

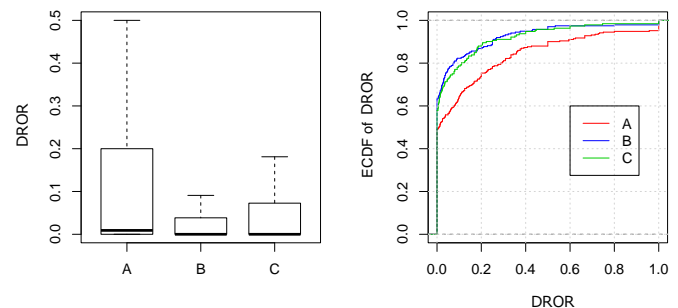


Figure 4. Boxplot and graph of ECDF of DROR in experiment 1

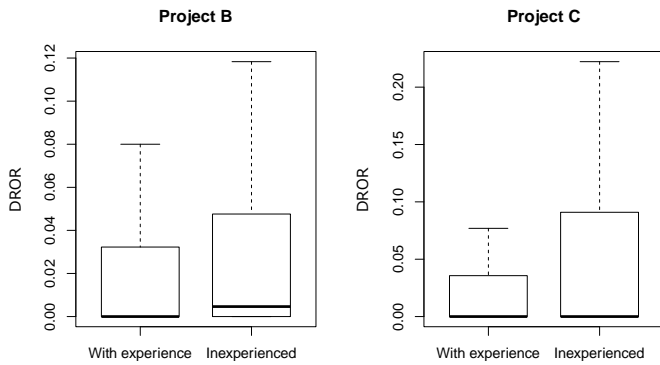


Figure 5. Boxplots of DROR in experiment 2

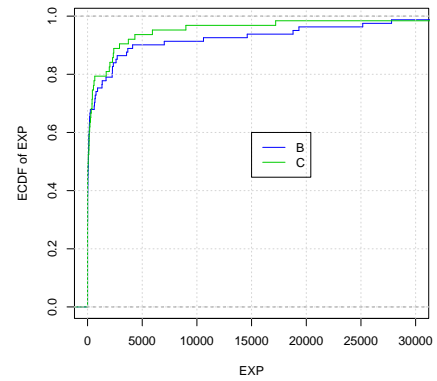


Figure 7. Graph of ECDF of EXP in experiment 3

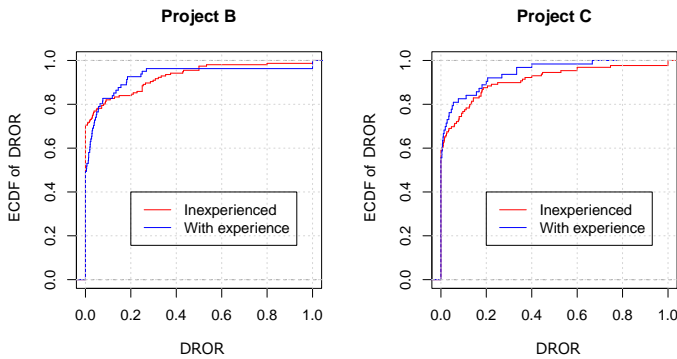


Figure 6. Graphs of ECDF of DROR in experiment 2

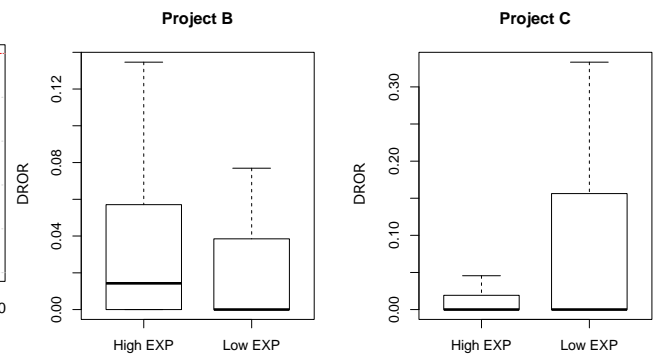


Figure 8. Boxplots of DROR in experiment 3

Figure 6 represents the ECDF graphs of DROR in evaluation experiment 2. The proportion of developers with experience and a DROR of 0.1 or less is higher than that in project B. On the other hand, the relation is opposite if the proportion is more than 0.1. In project C, regardless of reading the vertical axis with any position of a horizontal axis fixed, Fig. 6 indicates that DROR's of developers with experience is lower than those without experience. Moreover, judging from ECDF of inexperienced developers in both of project B and C, their DROR differs by project.

The above results show that the DROR of experienced developers is lower than that of inexperienced developers. The difference between the groups depends on the inexperienced developers and varies by project.

RQ3: Is there difference in DROR between developers based on experience in a similar project?

Figure 7 shows a graph of ECDF of EXP in evaluation experiment 3. EXP depends greatly on the number of changing files in a project due to its definition. Thus, ECDF of EXP differs by project, indicating that EXP cannot be used to compare traversing projects of developers.

Figure 8 shows the boxplots of the DROR in evaluation experiment 3, while Fig. 9 graphs ECDF of the DROR. With regard to project B, the width of the boxplot of developers with a high EXP is wider than those with a low EXP (Fig. 8). In addition, more than 60 percent of developers with high EXP in project B have greater than 0 DRORs (Fig. 9). These results suggest that other factors, which cannot be measured in terms of EXP, lead to defects. On the other hand, there is a gap between developers with high EXP and those with low EXP. This

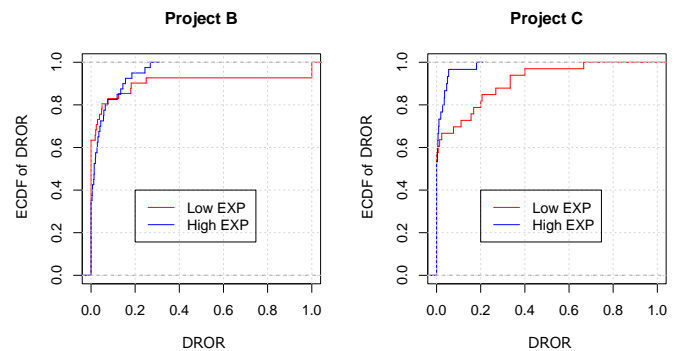


Figure 9. Graphs of ECDF of DROR in experiment 3

result suggests that when a developer involved in one project decides to engage in a similar one, his DROR should be reduced.

It remains to be seen if there is the difference of DROR's between developers with different experience levels. However, as the number of projects increases, our analysis method and EXP should be a useful metric to evaluate developers.

C. Threats to validity.

Internal validity:

This research focused on projects B and C, which were derived from the development of project A. Except for the notation variability of the absolute path among projects, if the absolute paths of a file in current development corresponded to that in past development, they were regarded as the same development function. Otherwise, they were viewed as quite different functions. This is a threat to internal validity. In the future, the influences of this assumption on this analysis method must be confirmed by comparing the similarity between

path names or function names inferred from path name, not the coincidence between absolute paths.

In addition, we determined the DROR based on the hypothesis that the developer who changed a file related to defect also induced the defect. As a result, some developers had a DROR of 1.0; in other words, some developers always caused defects. Although they worked as debuggers in actual development, this may affect the experimental results. This is also a threat to internal validity. In the future, who induced a defect must be more accurately identified by applying the SZZ algorithm proposed by J. Sliwerski et al. [12]. This algorithm infers commits, which brought about defects from *diff* and *annotate* commands of the version control system. By the additional investigation, we will clarify the correctness and limitation of the above-mentioned hypothesis in detail.

External validity:

In this experiment, we used Perforce as the version control system and Prisma as the bug tracking system. This is a threat to external validity. However, the analysis method of this paper is not designed for this experiment. So it may be effective in the same way for the domain that uses both a version control system and a bug tracking system. In the future, the efficiency of other domains and companies that handle version control systems and bug tracking systems must be verified.

VI. SUMMARY OF FINDINGS AND USAGE

Summary of findings are: 1) DROR cannot be used to compare different projects in the same organization. 2) There is certainly a difference in DROR's between experienced and inexperienced developers. 3) EXP should be a useful model to evaluate developers as the number of projects increases.

If the next development project is similar to past projects, our method provides useful information to improve personnel assignments. It can arrange the system so that experienced developers guide inexperienced ones as they work on development together. This should improve the quality of the software developed in the next project.

VII. CONCLUSION AND FUTURE WORK

To examine the tendency of DROR of developers based on development experience, we propose an analysis method and two models, which evaluate developers across multiple projects using their records in the same organization. The research found that despite being the same domain, comparing projects directly is not useful and that DROR of the developers with experience is lower than those without experience. Although it is unclear where there is a difference in the defect flow rates between developers with much and some experience, our proposed analysis model, EXP, should help evaluate developers for future projects.

As a future work, we will investigate files or absolute paths changed by developers who had high DROR despite having a lot of experience. If this is understood, it might be possible to evaluate the difficulty of functions to be developed, which may improve the precision of EXP. Moreover, we would like to try to discuss relations among defect removal overhead, defect inflow (how many

defects a developer introduced in files he changed) and defect removal efficiency (how many fixes a developer processed in all defects) to improve the precision when evaluating developer.

ACKNOWLEDGMENT

Our thanks go to anonymous reviewers who gave us a lot of valuable comments to improve this paper.

REFERENCES

- [1] C. Bird, N. Nagappan, B. Murphy et al., "Don't Touch My Code! Examining the Effects of Ownership on Software Quality," *ESEC/FSE '11 Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp.4-14, 2011.
- [2] M. Conway, "How Do Committees Invent?," *Datamation*, vol.14, no.4, pp.28-31, 1968.
- [3] P. Donzelli, R. "Handling the knowledge acquired during the requirements engineering process - a case study -," *SEKE '02 Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pp. 673-679, 2002.
- [4] J. Eyolfson, L. Tan, P. Lam, "Do Time of Day and Developer Experience Affect Commit Bugginess?," *MSR '11 Proceedings of the 8th Working Conference on Mining Software Repositories*, pp. 153-162, 2011.
- [5] F. Fagerholm, M. Ikonen, P.Kettunen et al., "How do Software Developers Experience Team Performance in Lean and Agile Environments?," *EASE '14 Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, No.7, 2014.
- [6] Y. Kamei, E. Shihab, B. Adams et al., "A Large-scale Empirical Study of Just-in-Time Quality Assurance," *IEEE Transactions on Software Engineering*, vol.39, no.6, pp. 757-773, 2013.
- [7] E. Kocaguneli, A. T. Misirli, B. Caglayan et al., "Experiences on Developer Participation and Effort Estimation," *SEAA 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp.419-422, 2011.
- [8] R. Latorre, "Effects of Developer Experience on Learning and Applying Unit Test-Driven Development," *IEEE Transactions on Software Engineering*, vol.40, No.4, pp. 381-195, 2014.
- [9] S. Matsumoto, Y. Kamei, A. Monden et al., "An Analysis of Developer Metrics for Fault Prediction," *PROMISE '10 Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, No.18, 2010.
- [10] A. Mockus, "Organizational Volatility and its Effects on Software Defects," *FSE '10 Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pp.117-126, 2010.
- [11] A. Mockus, D. M.weiss, "Predicting Risk of Software Changes," *Bell Labs Technical Journal*, Vol.5, No.2, pp.169-180, 2000.
- [12] N. Nagappan, B. Murphy, and V. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study," *ICSE '08 Proceedings of the 30th international conference on Software engineering*, pp.521-530, 2008.
- [13] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Programmer-based Fault Prediction," *PROMISE '10 Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, No.19, 2010.
- [14] S. Sato, H. Washizaki, Y. Fukazawa, S. Inoue, H. Ono, Y. Hanai and M. Yamamoto, et al., "Effects of Organizational Changes on Product Metrics and Defects," *APSEC 2013 20th Asia-Pacific Software Engineering Conference*, vol.1, pp.132-139, 2013.
- [15] E. Shihab, A. E. Hassan, B. Adams et al., "An Industrial Study on the Risk of Software Changes," *FSE '12 Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, No.62, 2012.
- [16] J. Sliwerski, T. Zimmermann, and A. Zeller, "When Do Changes Induce Fixes?" *MSR '05 Proceedings of the 2005 international workshop on Mining software repositories*, pp.1-5, 2005.
- [17] Y .Wu, Y. Yang, Y.Zhao et al., "The influence of developer quality metrics for fault prediction," *SERE 2014 Eighth International Conference on Software Security and Reliability*, pp.11-19, 2014.

Model Comparison: a Systematic Mapping Study

Lucian José Gonçalves, Kleinner Farias, Murillo Scholl, Maurício Veronez
PIPCA, University of Vale do Rio dos Sinos (Unisinos)
São Leopoldo, RS, Brazil
lucianjosegoncales@gmail.com,
kleinnerfarias@gmail.com, murillosholl@hotmail.com,
veronez@unisinos.br

Toacy Oliveira
PESC/COPPE, Federal University of Rio de Janeiro
(UFRJ)
Rio de Janeiro, RJ, Brazil
toacy@cos.ufrj.br

Abstract— Context: Model comparison plays a central role in many software engineering activities. However, a comprehensive understanding about the state-of-art is still required. **Goal:** This paper, therefore, aims at classifying, identifying publication fora, and performing thematic analysis of the current literature in model comparison for creating an extensive and detailed understanding about this area, thereby determining gaps by graphing and pinpointing in which research areas and for which study types a shortage of publications still exists. **Method:** We have conducted a systematic mapping study to scrutinize those contributions produced over time, which research topics have most investigated, and which research methods that have been applied. For this, we have followed well-established empirical guidelines to define and apply a systematic mapping study. **Results:** The results are: (1) majority of studies (14 out of 40) provide generic model comparison techniques, rather than comparison techniques for UML diagrams; (2) a categorization and quantification of the current studies in a variety of dimensions; and (3) an overview of current research topics and trends.

Keywords-component; model comparison, model matching, mapping study, model similarity.

I. INTRODUCTION

Model Driven-Engineering (MDE) is a model-centric approach where developers focus on elaborating, maintaining, and evolving design models in different levels [1][2]. In this context, model comparison plays a central role in many MDE activities as software models are in constant changes i.e., deletions, additions, and updates, are frequently occurring in such artifacts [2][3]. The goal behind managing models is to abstract software development process, i.e., guided by ideas and stakeholders goals instead typing a thousand of line codes. Academia has provided comparison techniques; at the same time, industry has provided robust tools. However, the resolution of model comparison did not reach an ideal scenario. Robust tools like IBM RSA [4], Epsilon [5] and MATA [6], still suffer from model comparison problems.

This is due their functionalities are far from providing a precise and large-scale computation in synchronizing and matching models. In [7][8], comparison problem is usually associated to graph isomorphism, well-know to be hard to resolve. Then, we can conjecture that current tools and comparison techniques do not solve the entire comparison problem yet, and then we still stand on the craftsmanship era.

We understand that a comprehensive understanding about the state-of-the-art is crucial for evolving the current comparison techniques.

This paper, therefore, aims at classifying, identifying publication fora, and performing thematic analysis of the current literature in model comparison for creating an extensive and detailed understanding of the state-of-the-art in this area. Moreover, we seek to determine gaps by graphing and pinpointing in which research areas and for which study types a shortage of publications still exists.

In this sense, we have conducted a systematic mapping study [9][10] to (1) scrutinize those contributions produced over time, and (2) characterize previously published model comparison approaches, i.e., which research topics have been most investigated, and which research methods that have been applied. For this, we have followed well-established empirical guidelines for defining and applying systematic mapping study (e.g., [10][11][12][13][14][15]). This method focuses on collecting statistical data related to a set of research questions, and provides a body of knowledge to future researches.

Our results show that approximately 34% of all studies (14 out of 40) provide generic model comparison techniques, rather than comparison techniques for UML diagrams [16]. Moreover, we categorize and quantify the current studies in a variety of dimensions, and give an overview of current research topics and trends. Finally, we have observed that existing literature has focused on providing repeated solutions for similar problems rather than on innovative approaches.

The remainder of this paper is organized as follows. Section II discusses the related work. Section III presents the SMS planning, i.e., the main steps for guiding the Systematic Mapping Study (SMS). Section IV presents the study results. Section V discusses the results by presenting a study map. Section VI presents some complementary results. Section VII shows the threats to validity. Finally, Section VIII presents some conclusions.

II. RELATED WORK

This section reports a series of studies, including surveys, mapping studies, and systematic reviews, which have previously reviewed the state-of-the-art in model comparison. To the best of our knowledge, this paper is the first to investigate the main research question proposed in Table 1.

There is a lack in the academia and industry for systematic and mapping studies that summarize model matching approaches. That is, there is no study showing a widespread view about model composition techniques or even providing a body of knowledge to future researchers. In [17], authors present an analysis about model comparison techniques showing its differences and trade-offs according the matching scenario, and in [18], authors present a survey about the state-of-the-art comparison approaches where they categorize approaches by type of models. Therefore, both studies do not focus in present a summary of studies through mapping study protocols.

Usually, current surveys indirectly address model comparison in the context of clone detection and model versioning, rather than systematically deal with comparison issues in the field of UML model composition, for example. In [19] the authors present a survey about tools that executes three-way merge inside the versioning control system. Still, they highlight that comparison task is essential to update models in repository. Alanen and Porres [20] present a description of three model-independent differentiation algorithms in the field of model versioning.

Given that model comparison is widely used, many approaches have been proposed, including UML and non-UML based ones. In [21], Salami and Ahmed describe the state-of-the-art works considering reuse of UML artifacts. Nevertheless, they cover only UML approaches. Selonen [22] presents a survey on model comparison approaches focused on UML models. Unfortunately, none of them provides a careful report classifying, identifying publication fora, and performing thematic analysis of the current literature in model comparison, hampering the creation of an extensive and detailed understanding about this area.

To sum up, there has been very limited empirical research reporting and characterizing the state-of-the-art of model comparison. More specifically, we have identified five key gaps in the current literature: (1) a lack of understanding about how model comparison has been investigated in the last years, and on which perspectives it has been done; (2) a gap to draw a “big picture” view of model comparison beyond UML, such as the degree of abstraction of comparison mechanisms; (3) there is no understanding as to what extent the comparison techniques are accurate, and which research methods have been used to investigate such techniques; (4) limited knowledge about which diagrams are supported by the comparison techniques, and which improvement points are more urged; and, finally, (5) an overview about how automated the model comparison techniques are.

III. SMS PLAN STUDY

This section describes the scope and essential steps for executing the systematic mapping study (SMS). Section III.A summarizes the researcher questions. Section III.B defines the strategy for the searching studies. Section III.C lists the inclusion and exclusion criteria for studies selection. Finally, Section III.D specifies the data extracted from selected studies.

A. Research Questions

Table I shows the research questions addressed in this study and their motivations. We seek to understand which diagrams the current model composition techniques are able to work with. To date, little is known about to what extent the existing techniques support to matching, or even computing the similarity between specific-types of design models. To explore these questions, we have found 2581 papers and realized a comprehensive and thorough analysis in 40¹. To carry out this in-depth investigation, we first had to define some search strategies for finding the papers.

TABLE I. RESEARCH QUESTIONS

Research Question	Motivation
RQ1: What are the types of diagrams addressed by comparison techniques?	Find out the types of diagrams that comparison techniques support, thereby revealing the diagrams that have been considered important as well as identify improvement points.
RQ2: What are the data structures commonly used in the comparison algorithms?	Pinpoint which data structures are used in the comparison algorithms.
RQ3: What are the types or categories used for evaluating diagrams in similarity approaches?	Understand the different aspects in required to evaluate diagrams.
RQ4: How fine-grained are the comparison techniques?	Grasp how accurate and detailed are the comparison techniques.
RQ5: What are the comparison types?	Explore if techniques are able to compare using different comparison strategies, thereby allowing to improve the precision of the similarity.
RQ6: Which empirical strategies are used to evaluate the comparison techniques?	Check the empirical strategies used to evaluate the comparison techniques.
RQ7: Is the approach automatic, semi-automatic or manual?	Investigate the level of automation used to compare models, thereby revealing the degree of human intervention required to compute the similarity score between two models.

B. Search Strategy

To search for the studies, we have defined terms to form Search Strings for performing searches in the main digital libraries. These strings were formulated following well-known empirical guidelines, (e.g., [10][11][12][15]), and followed a

TABLE II. SEARCH STRING (SS)

Major Terms	Synonym Terms
Diagrams	design OR model OR design OR structure
Comparison	match OR matching OR differencing OR similarity

five-step process to define the search terms as follows: (1) define the major keys; (2) identify alternative words,

¹<http://www.kleinnerfarias.com/publications/conference/seke2015>

synonyms or related terms to major keywords; (3) verify if the major keywords are contained in articles of the research category; (4) associate synonyms, alternative words or terms related to the main keywords with the Boolean “OR”; and (5) relate the major terms with Boolean “AND”.

The major keywords are “Diagram” and “Comparison”. Table II shows the synonyms and related words to major terms. We developed various combinations of Search Strings. However, we presented the substring that returned the most accurate results in search engines:

((Diagram OR Design OR Model OR Structure) AND (comparison OR matching OR differencing OR match))

The search string above was used in the major search engines for academic studies of the Internet: IEEE Digital Library, Science Direct, Digital ACM Library, Scopus, Google Scholar and Springer Link.

C. Selection: inclusion and exclusion criteria

We have used the following criteria to include the primary studies. First, search was limited to studies published in electronic digital libraries from newspapers or journals, educational institutions, international conferences, Master and PhD thesis. Secondly, we only considered approaches written in English. Thirdly, there has been no restriction on the publication year of studies until November 2014. Finally, papers which propose model comparison.

For approach exclusion, we have applied the following criteria: (1) papers and studies which not focus on model comparison; (2) duplicated studies returned by different search engines; and (3) papers and works that focus in low-level comparison (XML, source code and text).

D. Extracted Data

The following text describes the collected data we have extracted from articles to a spreadsheet and used it for summarizing the state-of-the-art model comparison techniques: (1) implicit data of inclusion and exclusion criteria: publication date, publication fora, and search engine; and (2) basic attributes of studies: main author and title; and finally (3) information related to research questions:

Diagrams (RQ1). The set of diagrams elicited from collected studies. They are accounted according these types of diagrams, including Component-and-Connector (CC), Generic (GD), Meta-Models (MM), Business Process Models (BPM), Use Case Diagram (UC), Class Diagram (CD), Sequence Diagram (SD), Activity Diagram (AD), Statechart Diagram (SCD), UML Profile (UP), and Any UML Diagram (AUD). Some diagrams are based in UML notation, but none of them was associated to a specific UML version.

Data Structures (RQ2). Basic data structures used by approaches and technologies.

Comparison aspects (RQ3). There is not a defined set of comparison aspects for model evaluation in the current literature. We have identified the following six comparison criteria in the works investigated: (1) *structure*, compare diagrams considering the modules and their relationships; (2) *syntactic*, compare taking into account the syntaxes of

diagrams; (3) *semantic*, compare diagrams considering the meanings of the differences; (4) *layout*, the comparison approaches aim at view issues; (5) *lexical*, implement a name-based model comparison; and (6) *multi-strategy*, the approaches combine at least two comparison strategies to improve the comparison results.

Granularity levels (RQ4). Granularity refers to the unit of conflicts, e.g., attributes of the input models, and depends on the diagram used and criteria evaluated (item 3) e.g., layout aspects on UML class diagram own it is specific attributes. Users can set the level of granularity according the desired scalability and user's convenience [23]. We categorized the model comparison in tree levels of granularity: (1) *coarse-grained*, only one attribute is analyzed to compute the elements differentiation, e.g., the element names only; (2) *partial*, a set of attributes that is analyzed, i.e., more than one element; and (3) *fine-grained*, use all the possible attributes for execute the diagrams differentiation;

Comparison Type (RQ5). Comparison techniques can find the commonalities and differences between models using different strategies. We have identified two types of model comparison: (1) *similarity*, the mechanism's goal is to identify the similarity returning values indicating how similar the elements between each other are; and (2) *matching*, the mechanisms return a set of matched elements.

Research method categories (RQ6). This is a question that provides a general view about the direction of the current studies, i.e., the kind of studies that academia have been producing. Given that there is a vast amount of works to be classified, we have used the categories proposed in [12] for classifying the selected papers: (1) *evaluation research* uses empirical strategies to evaluate proposed works; (2) *solution proposal* proposes a solution based on new or previous approaches; (3) *validation research* used for evaluating techniques, which have not been widely adopted in industry; (4) *philosophical papers* proposes new and revolutionary research to address some aspects of model comparison; and (5) *opinion papers*, studies that remain the discussion about the author's point of approaches arguing to resolve the tackled problem based in previously personal experiences.

Autonomous level (RQ7). In order to know the kind of automation support that algorithms provide to users, we have investigated the current works from tree perspectives: (1) *automatic*, it does not require any human interaction; and (2) *semi-automatic*, it requires users specifying configuration parameters before differentiation execution. Those approaches need user intervention for handling evaluation procedure, and (3) *manual*, a list with strategy steps or good practices for conducting the diagram comparison.

Technique description (RQ8). We have also observed how authors represent the comparison algorithms one. We generalized the following studies according the selected studies content: (1) *pseudo-code*, the approach shows the algorithm in a generic formalism, i.e. language-independent; (2) *textual*: authors to explain how the approach works by plain text; and (3) *other*, language-dependent and formalism representations (programming languages, modeling representations, etc.).

IV. EXECUTION

We have adopted the following four steps to select studies. The list below describes the sub-phases used to find studies, and Table III shows the results obtained in each sub-phase.

- Step 1. *First results (SP1)*: find electronic papers using the substring, according Section III.B.
- Step 2. *Duplicates Removed (SP2)*: remove repeated studies.
- Step 3. *Pre Selection (SP3)*: remove papers that do not match in established requirements and research questions.
- Step 4. *Selected Studies (SP4)*: we analyzed all selected studies in the previous step and applied the exclusion criteria aforementioned.

TABLE III. STUDIES OBTAINED IN EACH STEP

Steps	IEEE	Scopus	Springer Link	Google Scholar	ACM	Science Direct	Total
SP1	270	461	891	427	49	483	2581
SP2	268	321	787	392	45	476	2289
SP3	41	49	87	93	20	9	299
SP4	7	2	2	23	6	0	40

V. STUDY RESULTS

This section presents the results for each research question as follows.

A. Diagrams Category (RQ1)

The majority of the works (14 approaches) focused on generic diagrams, i.e, those diagrams that are more abstract and consider similar attributes to compare the diagrams. Model comparison plays an important role inside MDE, where the capability in comparing many kinds of models are required. In addition, a recent study concluded most developers and large companies think UML complex leading the use of more abstract and alternative models [24]. This is the explanation more accepted about the high quantity of approaches focusing on resolving generic diagram comparison. On the other hand, the class diagram is the most common UML diagram investigated (12 approaches); similar to previous studies [25][26] in the field of software modeling highlighting UML class diagram as one of UML most used in practice. Although UML is considered de fact standard modeling language [26], we have observed the number of non-UML-based comparison techniques (48%) outnumbers the UML-based ones (53%).

B. Data Structures Used (RQ2)

Majority of the approaches (58%) use graphs for model comparison. The tree data structure is the second most used (14%) by authors. The minority (3%) implemented semantic similarities, and other authors (25%) utilize other simpler data structures.

C. Comparison Aspects (RQ3)

The results about comparison aspects revealed 21 approaches focusing in structural comparison aspects. After, 11 papers focused on Multi-Strategy comparison, i.e., comparison mechanisms using more than one strategy. Moreover, three

papers evaluated the semantic aspects, and other 3 studies focused in lexical comparison, i.e., only evaluating the differences between words and one algorithm evaluated the layout characteristic, and one article focusing in syntactic aspect. The greater part of approaches evaluates one aspect, i.e., 29 approaches, and the remaining studies evaluated more than one aspect.

D. Granularity (RQ4)

Most approaches (29) proposed coarse-grained approach, i.e., they evaluate only one level of abstraction. Section V.A pointed that most approaches focused on generic diagrams. This leads model comparison algorithms considering main attributes for evaluating models. Other only three approaches executed a fine-grained evaluation, i.e., a detailed attributes coverage.

E. Comparison Type (RQ5)

We have classified the comparison mechanisms in two categories: (1) similarity, focusing on math similarity; and (2) matching, approaches performing a mapping of model elements. This definition is located in the Section III.D. The matching approach (28 of 40 studies) is the most used, followed by similarity approach (12 of 40 studies).

F. Research Method (RQ6)

The majority of approaches (77,5%, 31 of 40 papers) are Proposal of Solution. Evaluation Research and Philosophical Papers has the same quantity (10%, 4 of 40 papers) and, the resting approaches (2,5%, 1 of 40 papers) rely on practical experience to develop a comparison approach. There are two main aspects to consider in this research question: (1) there are new emerging approaches for model comparison; and (2) the approaches do not complement each other. The results show that the literature has recurrently proposed and discussed new comparison techniques. During the whole selection steps, we did not find any opinion paper about model comparison.

G. Autonomously Level (RQ7)

Most algorithms conduct the model comparison process autonomously (67,5%, 27 of 40 papers), followed by semi-automatic process (30%, 12 of 40 papers) where users must set some adjustments during the comparison and, only one approach (2,5%, 1 of 40 studies) discusses how persons manually match UML diagrams. From the results is possible to perceive a strong tendency for producing automatic approaches. We understand the focus in the process of diagram comparison is to avoid users wasting time with specific configurations.

VI. COMPLEMENTARY RESULTS

Table IV shows a list that contains a rank of those publication fora that contains more papers focusing on comparison of diagrams. Table represents 27.5% of all papers analyzed (11 of 40 articles). Conferences/Journals with one article did not appeared in this Table. Figure 1 presents that the frequency of publications was higher in 2008 and 2011. This period was responsible for producing 19 studies, i.e., a higher quantity production than other periods from 2003 to 2007, and from 2012 until 2014.

TABLE IV. QUANTITY OF PAPERS PER EVENT/JOURNAL

Publication Place	Quantity of approaches	Percentage
IEEE/ACM International Conference on Automated Software Engineering (ASE)	4	10%
IEEE Transactions on software Engineering	3	8%
European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)	2	5%
International Conference on Software Maintenance (ICSM)	2	5%

The production was the most unproductive in 2009 (just 1 article produced). In a general overview over the chart is possible to perceive a frequent times of rise and falls in publication numbers. Moreover, the average of produced studies is low (by about 3,33 studies per year).

Figure 1. Publications by year

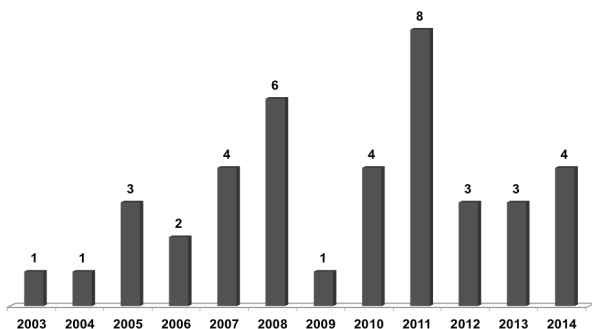


Table V shows a rank of all authors according their productivity in relation to matching and similarity approaches publications. The number of papers was accounted taking in consideration the author's first name in studies. Zhenchang Xing is the author that most produced comparison techniques (8%, 3 from 40 studies). After six authors produced two papers, and others authors produced each one approach.

TABLE V. AUTHORS PUBLICATION QUANTITY RANKING

Author	Quantity of Approaches	Percentage
Zhenchang Xing	3	8%
Christian Gerth	2	5%
Hamza Onoruoiza Salami	2	5%
Kleinner Oliveira	2	5%
Mark van den Brand	2	5%
Segla Kpodjedo	2	5%
Shiva Nejati	2	5%

The following discussion describe the analysis of the results (Figure 2) for the combined research questions illustrated in a bubble plots. This method gives a map and provides a general overview of what academia published. Combining the results of RQ1, RQ6, and year, we obtained the mapping of the

evolution through years of the number of type diagrams and the research methods used. The results show that the proposal of solution is the research type that have been more adopted by academia to present comparison approaches over the last 11 years. In 2008, all 6 papers were proposal of solution, where 3 focused the comparison approach on class diagrams and respectively one approach for generic, meta-model and UML profiles. Generic diagram received more attention by three years in academia. Firstly, in 2007, two proposals were published concerning generic comparison of diagrams. This happened after by about six years from the Model-Driven Architecture (MDA) has been formally proposed, and according our results, after four years, the beginning of the model comparison studies in the academia. Second, in 2010, all four proposals of solutions were concerned in generic diagram comparison; a year before (2009) only one study was published with focus on class diagrams, i.e., the most unproductive year. Finally, in 2011, generic digram was the center of attention by the second year consecutive.

VII. THREATS TO VALIDITY

We follow the systematic mapping study methodology for the execution of this research. This method provides protocols to extract data in order to guarantee detailed results of the state of the art. For this we defined the search strings and research questions. However, some factors may threat the validity of the study: (1) difficulty to relate all works to the topic due the constant changes in publications; and (2) the conduction of data extraction of the papers, such as (1) the search string we used has the main terms such as “model” and “matching”. However, “matching” and its synonyms (comparison, similarity, etc.) are generic and this string retrieved broad results; (2) the inclusion of thesis and dissertations published on-line that are not peer revied and, (3) the limitation to the main six search engines defined in the SMS planning.

VIII. CONCLUSION

This paper identified and classified publication fora, and performed thematic analysis of the existing literature in model comparison, thereby providing an in-depth understanding about the model comparison area. In addition, it addressed this gap by describing and pinpointing in which field and for which research topics a shortage of publications still exits.

We have observed that the most studies have concentrated more effort on producing generic comparison techniques, rather than on providing specific ones, e.g., techniques for comparing UML models. This can be explained by three reasons. First, there is not a widely-adopted modeling language in industry. Second, given the wide variations of modelling notations and diagrams types, it would be challenging to provide an approach that can have a broad adoption. Third, model comparison is not a trivial task to deal with. Rather, it may be still characterized a time-consuming and error-prone task. Finally, we also hope that this work represents a first step in a more ambitious agenda on providing a better support researchers and practitioners to compare models. In addition, we hope that the issues outlined throughout the paper may encourage other researchers to extend our study.

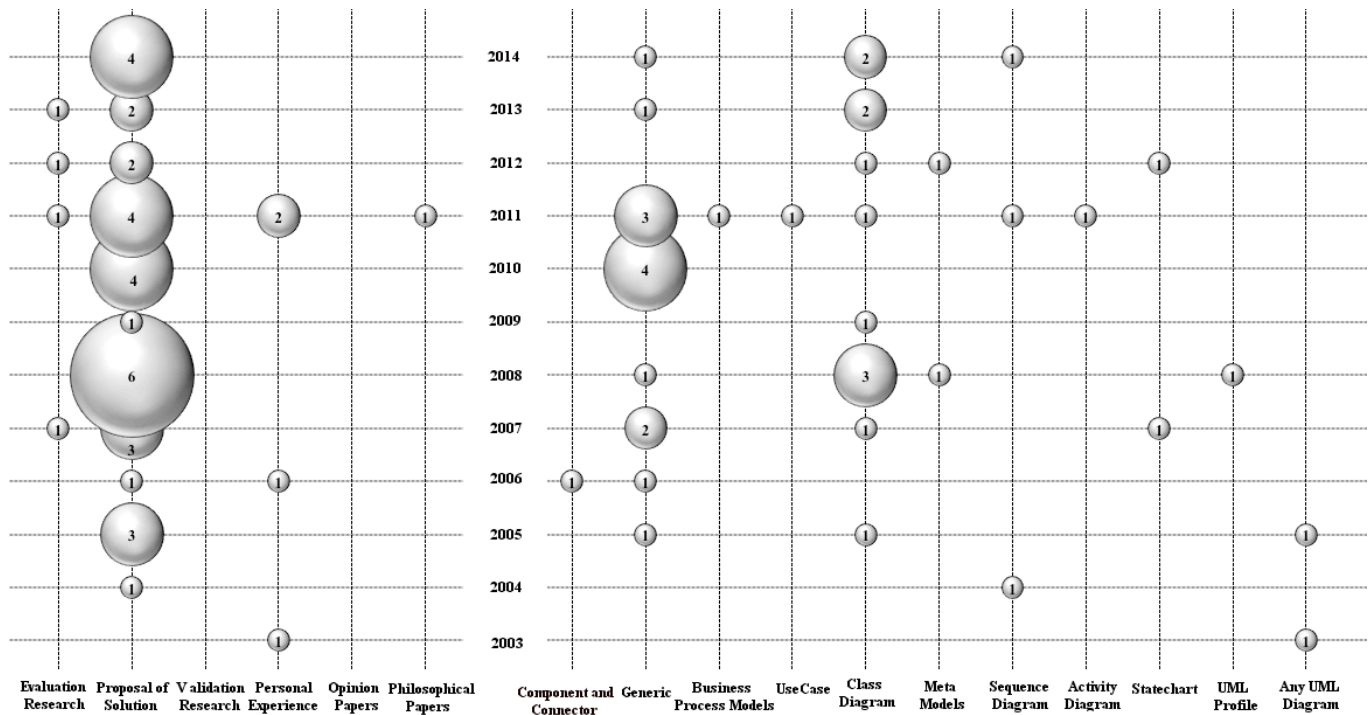


Figure 2. Publications by year

ACKNOWLEDGMENT

This work was funded by Universal project – CNPq (grant number 480468/2013-3).

REFERENCES

- [1] S. Kent, "Model-driven engineering," In: 3rd Int. Conf. on Integrated Formal Methods (IFM '02), pages 286-298, 2002.
- [2] A. Sarma, D. Redmiles, A. Van der Hoek, "Palantir: early detection of development conflicts arising from parallel code changes," *Software Engineering, IEEE Transactions on*, vol.38, no.4, pp.889,908, July-Aug. 2012.
- [3] Wieland *et al.* "Turning conflicts into collaboration," *Computer Supported Cooperative Work*, pages 181-240, 2013.
- [4] IBM, IBM Rational Software Architecture. <http://www.ibm.com/developerworks/downloads/r/architect/index.html>, accessed in 2015.
- [5] Epsilon, <https://www.eclipse.org/epsilon/>, accessed in 2015.
- [6] J. Whittle, P. Jayaraman, Synthesizing hierarchical state machines from expressive scenario descriptions, *ACM Trans. Softw. Eng. Methodol.*, vol. 19, no. 8, February 2010.
- [7] K. Voigt, "Structural graph-based metamodel matching," PhD thesis, University of Desden, 2011.
- [8] S. Abbas, H. Seba, "A module-based approach for structural matching of process models," 5th Int. Conf. on Service-Oriented Computing and Applications, pages 17-19, 2012.
- [9] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, "Systematic mapping studies in software engineering," 12th Int. Conf. on Evaluation and Assessment in Software Engineering, UK, pp. 68-77, 2008.
- [10] D. Budgen *et al.*, "Using mapping studies in software engineering," *Proceedings of PPIG*, vol. 8, pp. 195-204, 2008.
- [11] B. Kitchenham, P. Brereton, D. Budgen, "The educational value of mapping studies of software engineering literature," 32nd Int. Conf. on Software Engineering, vol. 1, New York, NY, USA, pp. 589-598, 2010.
- [12] B. Kitchenham, D. Budgen, O. Brereton, "Using mapping studies as the basis for further research - A participant-observer case study," *Inf. Softw. Technology*, pp. 638-651, 2011.
- [13] D. Torre, Y. Labiche, M. Genero, "UML consistency rules: a systematic mapping study," 18th Int. Conf. on Evaluation and Assessment in Software Engineering, 2014.
- [14] N. Asoudeh, Y. Labiche, "Requirement-based software testing with the UML: a systematic mapping study," 7th Int. Conf. on Software Engineering Advances, 2012.
- [15] Wohlin *et al.*, "Experimentation in Software Engineering", Springer, Heidelberg, Berlin, Germany, 2012.
- [16] OMG, UML metamodel: superstructure specification, <http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF>, 2015.
- [17] D. Kolovos, D. Ruscio, A. Pierantonio, R. Paige, "Different models for model matching: an analysis of approaches to support model differencing", Workshop on Comparison and Versioning of Software Models(CVSM '09), pages 1-6, 2007.
- [18] M. Stephan, J. Cordy, "A survey of model comparison approaches and applications," in International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp.265-277, 2013.
- [19] K. Altmanninger, S. Martina, M. Wimmer, "A survey on model versioning approaches," *Int. Journal of Web Information Systems*, pages 271-304, 2009.
- [20] M. Alanen, I. Porres, "Difference and union of models," UML conference, pages 2-17, 2003.
- [21] H. Salami, M. Ahmed, "UML artifacts reuse: state of the art," 2014.
- [22] P. Selonen, "A review of UML model comparison approaches," Nordic Workshop on Model Driven Engineering, 2007.
- [23] B. Bruegge, A. de Lucia, F. Fasano, G. Tortora, "Supporting distributed software development with fine-grained artefact management," 6th Int. Conf. on Global Software Engineering, pages 213-222, 2006.
- [24] M. Petre, "UML in practice," International Conference on Software Engineering, Piscataway, NJ, USA, pages 722-731, 2013.
- [25] A. Nugroho, M. Chaudron. "A survey into the rigor of UML use and its perceived impact on quality and productivity," In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '08). ACM, New York, pp. 90-99, 2008.
- [26] M. Chaudron, W. Heijstek, and A. Nugroho. "How effective is UML modeling ?" *Software System Modeling*. Vol. 11, no. 4 , pp. 571-580, 2012.

Exploring SOA Pattern Performance using Coupled Transformations and Performance Models

Nariman Mani, Dorina C. Petriu, Murray Woodside
Department of Systems and Computer Engineering, Carleton University
Ottawa, Ontario, Canada
{nmani | petriu | cmw}@sce.carleton.ca

Abstract—Service Oriented Architecture (SOA) patterns can be applied to improve different qualities of SOA designs. The performance impact of a pattern (improvement or degradation) may affect its use, so we assess its impact by automatically generated performance models for the original design and for each candidate pattern and pattern variation. This paper proposes a technique to incrementally propagate the changes from the software to the performance model. The technique formally records the refactoring of the design model when applying a pattern, and uses this record to generate a coupled transformation of the performance model. The SOA design is modeled in UML extended with two profiles, SoaML and MARTE; the patterns are specified using Role Based Modeling and the performance model is expressed in Layered Queuing Networks. Application of the process, and pattern performance exploration, is demonstrated on a case study.

Keywords- *Software performance model, service oriented systems, SOA pattern, coupled transformation, LQN*

I. INTRODUCTION

In designing SOA (Service Oriented Architecture) systems, SOA patterns [1] are proposed as generic solutions to problems in the architecture, design and implementation. The patterns may have a substantial impact on performance, and we wish to evaluate this with a performance model (PModel) generated automatically from a software design model (SModel) and the pattern description. The baseline PModel may be created by an automated transformation as in PUMA (Performance from Unified Model Analysis) [2]. The automated refactoring of the PModel to reflect application of a pattern, using coupled transformations of the SModel and the PModel, is the subject of this work. Automating the refactoring makes it easier to consider the performance issues, and to rapidly consider a (possibly large) set of variations on a pattern. It also reveals the causal connections between the design changes and the performance issues, which may be of value to the designer. Manually refactoring the SModel and then regenerating the PModel using PUMA is a viable alternative but may suffer from inconsistency in the refactoring. In [3] we studied the impact of SModel changes to PModel due to application of SOA design patterns.

This research describes a coupled transformation technique to incrementally propagate design changes to the PModel by: (A) definition of the pattern using a role-based modeling technique; (B) formal recording the SOA design refactoring;

(C) automatic derivation of the corresponding performance model changes; (D) application of the changes to the PModel. This paper describes (A) – (C) but does not address the implementation of the transformation in step (D). The SOA SModel is captured in UML with the OMG profiles SoaML (Service Oriented Architecture Modeling Language) [4] and MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [5] for performance information. The Role Based Modeling Language (RBML) [6] is used to formally define each SOA design pattern in terms of first, the set of SModel elements that represent the *problem* addressed by the pattern and second, those that constitute the *solution*. The novel contributions of this work are the coupled transformation in Section VI, and the process (systematic and automatic) that supports its use, including the formal recording of the changes for SModel refactoring.

The paper is organized as follows: Section II presents related work, Section III surveys the approach; Section IV describes the models; Section V describes the SModel transformation rules; Section VI presents the coupled transformation; Section VII describes a case study. Finally Section VIII concludes the paper.

II. RELATED WORK

The relationship of PModels to SModels, and the derivation of one from the other, is the subject of considerable work, including diverse target PModel types such as Queuing Networks, Layered Queuing Networks (LQNs) [7] and Stochastic Petri nets [8]. The general approach of PUMA integrates diverse types of PModel and SModel [2]. This work uses it with UML SModels (for the SOA designs) annotated with MARTE, and LQN PModels. The SModel-to-PModel mapping of [9] is extended here to support the coupling of the refactoring transformations.

The impact of design patterns on software performance has been studied only indirectly, through the concept of performance anti-patterns, introduced in [10]. Anti-patterns are defined as common design errors that cause undesirable results. An approach based on anti-patterns for identifying performance problems and removing them is described in [11]. An OCL query is created to identify each anti-pattern and applied to the design model. The anti-pattern removal is special for each anti-pattern and is not automated.

Xu [12] described a rule-based system which discovered performance problems and automatically improved the

design as represented by the PModel. However the rules are slightly different from patterns or anti-patterns and the changes were not propagated automatically to the SModel.

III. PROCESS OVERVIEW

The overall process is shown in Figure 1. This paper describes stages B and C, shown in grey. The inputs include a SOA SModel (top left), and a library of pattern definitions with formal roles (bottom left). The designer steps are given on the left and the automated steps on the right side.

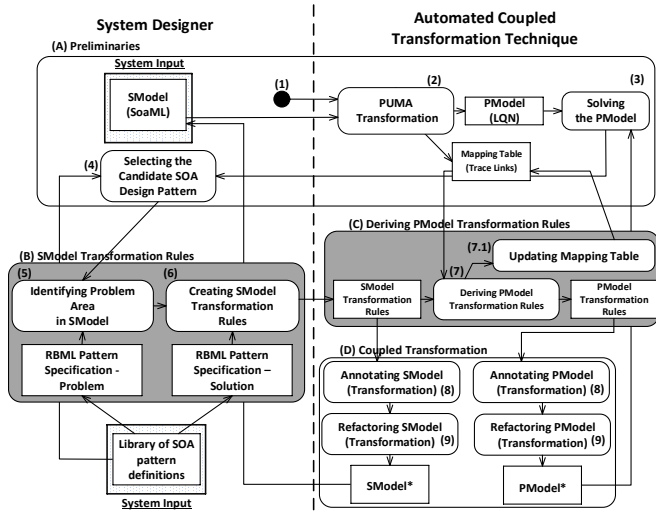


Figure 1: Proposed Approach Overview

The designer steps are supported by tools that have been implemented in this work. There are four stages:

A) Preliminaries: This stage uses the SModel to create the base PModel using PUMA, and creates the SModel/PModel mapping table. Pattern application begins at step (4), where the designer selects a candidate pattern for its own reasons (e.g. maintainability).

B) Model Transformation Rules: The selected pattern is specified using RBML. The designer indicates where the pattern is applied by binding pattern roles to entities in the SModel (step (5)) and then specifies SModel transformation rules that will satisfy the solution specification (step (6)).

C) Deriving the PModel Transformation Rules: Using the mapping table from (A) and the SModel transformation rules from (B), the PModel transformation rules are derived automatically.

D) Coupled Transformations: Both sets of transformation rules are executed via coupled transformations to refactor the SModel and PModel into SModel* and PModel*, respectively. The PModel* results can be used to select the pattern to be applied. Therefore, Stages B, C and D may be repeated until the designer gets the desired results.

IV. MODELS

A. SOA Models

From the range of diagrams used to model SOA systems, we use the Business Processes Model (BPM) for behavior

and the Service Architecture Model (SEAM) for structure and contracts, together with a UML deployment diagram. Figure 2 shows examples.

The BPM is specified as a UML activity diagram (Figure 2.B). Service invocations are modeled as operation calls, using three types of UML actions: a *CallOperationAction* transmits a request to the target and waits for the reply via its input/output pins; an *AcceptCallAction* is an accept event action waiting for the arrival of a request; and a *ReplyAction* returns the reply values to the caller. The called operation appears in parentheses after the action name as “(class-name::operation-name)”. We assume all BPM edges between ActivityPartitions are between these three Action types and represent calling interactions.

Performance information by MARTE annotations are given in shaded notes. They describe the behavior as a sequence of steps «PaStep» with a workload attached to the first step («GaWorloadEvent»). «PaStep» has attributes *hostDemand* (the required CPU time), *rep* (the mean repetitions) and *prob* (its probability if it is an optional step). The workload «GaWorloadEvent» defines a population of *Nusers* users, each with a thinking time *ThinkTime* defined by MARTE variables. Concurrent runtime instances «PaRunInstance» are identified with swimlane roles.

The SEAM is specified as a UML collaboration diagram (Figure 2.A) with service participants and contracts (stereotyped «Participant» and «ServiceContract» respectively; these are not from MARTE but are specific to this process). Each participant plays a role of Provider or Consumer with respect to a contract. Participants correspond to pools, participants and swimlanes in the BPM.

Deployment is also defined, as in Figure 2.C. Processing nodes are stereotyped «GaExecHost» and communication network nodes are stereotyped «GaCommHost», with attributes for processing capacity, message latency and communication overheads.

B. Performance Models

PModels are expressed in an extended queueing notation called Layered Queuing Networks (LQNs) [2], selected because of its close coupling to the high-level software architecture. An LQN estimates waiting for service due to contention for host processors and software servers, and provides response time and capacity measures.

Figure 2.D shows the LQN model for the example. For each service there is a task, shown as a bold rectangle, and for each of its operations (contracts) there is an entry, shown as an attached rectangle. The task has a parameter for its multiplicity or thread pool size (e.g. {‘1’}). Each entry has a parameter for its host CPU demand, equal to the total *hostDemand* of the set of «PaSteps» for the same operation in the SModel. Calls from one entry to another are indicated by arrows between entries (a solid arrowhead indicates a synchronous call for which the reply is implicit, while an open arrowhead indicates an asynchronous call). The arrow is annotated by the number of calls per invocation of the sender. For deployment, an LQN host node is indicated by a round node associated to each task.

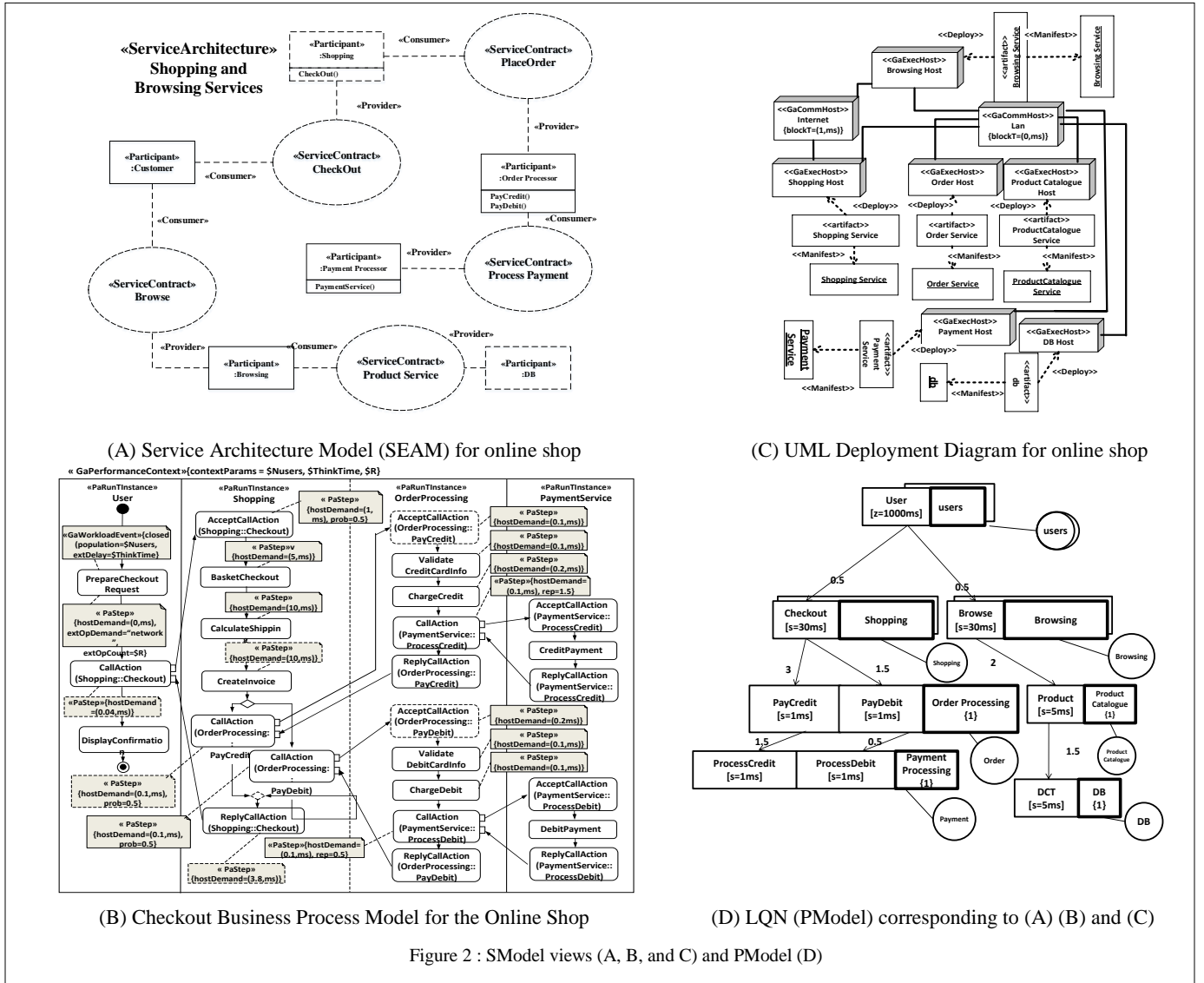


Figure 2 : SModel views (A, B, and C) and PModel (D)

C. SModel to PModel Mapping Table

When the PModel is derived from the SModel using the PUMA [2] process, the mapping between the corresponding elements of the two models is recorded as described in [9], extended to identify a set of Actions initiated by a Call (an ActivitySet), and pairs of Call and Reply Actions. There are three mapping sub-tables, for StructuralElements, Calls, and Attributes. Each row in a table represents a link between an SModel element or set and a corresponding PModel element (because the PModel is more abstract, one element may correspond to a set of SMEs). Table 1 shows a few of the traceability links for the example in Figure 2.

D. Role-Based Models for SOA Patterns

To formalize the definition of SOA design patterns without resorting to a new language, we use Role-Based Modeling RBML [6], where the pattern is expressed with generic roles acting as formal parameters which must be

bound to actual parameters from the application context to which the pattern is applied.

Table 1: Partial Mapping Table between SModel and PModel for Shopping and Browsing

Sub-table (A) StructuralElements Trace Links		
Link	Set of SModel Elements	PModel Element
DTL3	Deployment Node: Order	LQN Host: Order
DTL2	Deployment Artifact: Browsing	LQN Task: Browsing
BTL1	ActivitySet: Checkout = {AcceptCall, BasketCheckout, CalculateShipping, CreateInvoice, CallOperation(OrderProcessing::PayCredit), CallOperation(OrderProcessing::PayDebit), Reply}	LQN Entry: Checkout
Sub-table (B) Calls Trace Links		
Link	Set of SModel Calls	PModel Call
BCTL1	Call from CallOperationAction(Shopping::Checkout) to AcceptCallAction(Shopping::Checkout) and the corresponding reply from ReplyAction(Shopping::Checkout) back to CallOperationAction(Shopping::Checkout)	LQN synchronous Call from Entry:User to Entry: Checkout

Three UML views are used for each pattern: **BPS** (Behavioral Pattern Specification) for behavior, corresponding to the BPM; **SPS** (Structural Pattern Specification) for structure, corresponding to the SEAM; and **DPS** (Deployment Pattern Specification), not described here due to space limitations. Each view has two specifications: *Pattern Problem* (the view before pattern application) and *Pattern Solution* (after application). Figure 3 shows the role-based specification for the Service Façade pattern (which is described in Section VII), with the *problem* on the left and the *solution* on the right. As in [6] the names of generic roles start with the character ‘|’.

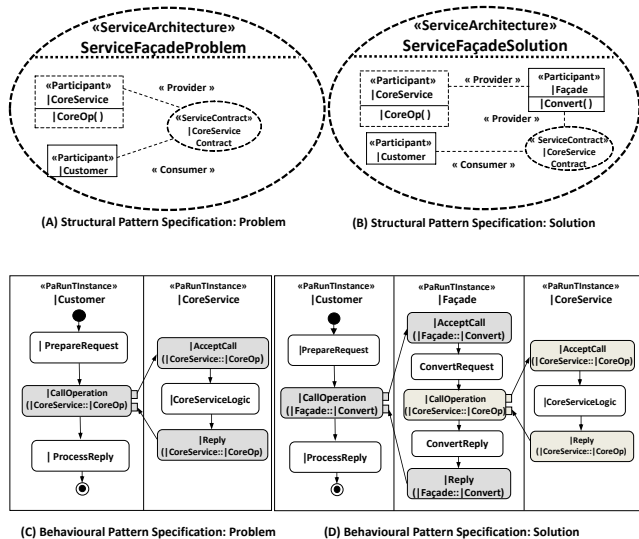


Figure 3: Service Façade pattern specification

V. SModel TRANSFORMATION RULES

The first step in applying a pattern is to identify the model elements to which it can be applied, based on the pattern problem. From these, particular elements are chosen as the *area of application* by binding them to roles in the RBM definition. It is not our goal to automate this process of selection and binding and then applying the pattern solution, but to make it systematic and to support it with a construction tool (as shown below in Figure 4).

A. Problem Identification and Role Binding

The designer chooses a pattern to apply and, using its RBM definition from a pattern library, binds the elements of its problem specification (SPS, BPS and DPS) to the elements of the SModel. An element can be bound if:

1. Its type matches the RBM element type.
2. It has all the attributes and operations defined by the RBM element.
3. Any constraints defined for the two matching elements are compatible (that is, the pattern does not impose additional constraints when applied to the SModel).
4. For the SModel behavioural view (BPM), the execution flow and the ActivityPartitions (swimlanes) must match.

Not every pattern specification element is defined as a role. Those which are not (e.g. Calls, Replies, Attributes) are also

bound, governed by the role bindings. These “derived bindings” may be determined by the binding of a single element (e.g. its Attributes) or from the bindings of multiple elements (e.g. a Connector between two elements). Some bindings for the BPM of the example (involving the pattern specification in Figure 3 and the SModel BPM in Figure 2.B) are given by the following pairs including a derived binding found between the RBM Call and SModel Call, which is implied by the binding of the core operation:

RBM Element	SModel Element
CallOperation(CoreService: CoreOp)	CallOperationAction(Shopping::Checkout)
AcceptCall(CoreService: CoreOp)	AcceptCallAction(Shopping::Checkout)
CoreServiceLogic	Sequence of all Actions in Shopping swimlane
(Derived Binding) Call from CallOperation(CoreService: CoreOp) to AcceptCall(CoreService: CoreOp)	Call from CallOperationAction (Shopping::Checkout) to AcceptCallAction (Shopping::Checkout)

B. Creating the SModel Transformation Rules

The designer creates the SModel transformation, (governed by the RBM bindings and the pattern problem and solution specifications) as a set of operations to add, delete, and modify model elements. An operation is defined for each element type (eg. addAssoc/deleteAssoc for adding/deleting associations). Depending on the element type it applies to, an operation is applied to the services and interactions of the SEAM and to the ActivityPartitions, Activities, Actions and ActivityEdges of the BPM. Transformation operations indicated by the designer are recorded using the tool shown in Figure 4 as follows:

- Remove elements that are present in the problem but not in the solution, by applying *delete* actions (such as *deleteParticipant* or *deleteAssoc*) to them
- Create new elements that are defined in the solution but are not present in the problem, by *add* actions (such as *addParticipant* or *addActivityPartition*),
- Modify elements present in both problem and solution, by *modify* actions (such as *modifyActionCall*).

SModel elements which are not in any of the above groups remained untouched. Figure 4 shows a screen shot of the tool support for the technique in this section with a set of operations recorded for the application of Service Façade pattern to the example in Figure 2, with the role bindings shown above.

VI. COUPLED TRANSFORMATION

A. Coupled PModel Refactoring Rules

This section describes the automated translation of the SModel transformation rules into PModel transformation rules, based on the mapping table described in Section IV.C. Each SModel transformation rule has an operation name and some arguments, which are processed as follows:

1. The operation name is translated into one or more PModel transformation operations. The action part of the name (add/delete/modify) is retained, and the operand-type part (e.g. Participant) is mapped according to the type correspondences of the Mapping Table. A partial list of these is:

SModel Type

- Participant
- ActivitySet
- Call/Reply pair of Actions
- Call (no Reply)
- ExecHost

PModel Type

- Task
- Entry
- Call (sync)
- Call (async)
- Host

Thus the SModel operation addParticipant is translated to addTask, and deleteActivitySet to deleteEntry.

- The arguments of the PModel operation (e.g. the entity or entities to be added, deleted, or modified) are translated from the arguments of the SModel operation using the correspondences in the Mapping Table. For “add” operations the name of the new PModel element is taken as the name of the corresponding SModel element.

For example, the SModel “addParticipant” operation is mapped to “addTask” in the PModel, and the “addParticipant” argument becomes the new task name.

Modifications to calls require special consideration in the translation. The SModel “modifyActionCall” operation changes a service invocation from a *CallOperationAction* to an *AcceptCallAction*. As this might apply to more than one call to the same *AcceptCallAction*, the mapping table is searched (by the MappingTableSearchByKey command) to identify all the PModel activities making the call. Then the operation is mapped to one or more “modifyActivity” operations in the PModel domain, to change all the calls.

Some of the PModel transformation rules derived from the Façade pattern (shown in Figure 4) are presented in Figure 5 as part of the screenshot from the implemented tool supporting coupled transformations.

B. Application of the PModel Rules

Briefly, the PModel transformation rules derived in Section VI.A are applied to the PModel in two steps. First the PModel is annotated with transformation directives indicating the changes, then the changes are applied by a transformation engine implemented using QVT [13] (Query, View, and Transformation, a OMG standard model transformation language) which processes the directives. The implementation of these two steps is not presented here.

VII. CASE STUDY

We suppose that a designer is assigned the task of re-designing the Shopping and Browsing SOA described earlier to support three different user access channels (mobile phone, desktop, kiosk, etc.) through a single multi-channel endpoint. Initially, the designer uses the SOA design pattern “Concurrent Contracts” [1] in which the multi-channel capability is implemented by providing separate shopping and browsing operations for each channel. Separate set of actions are created inside the shopping swimlane (see Figure 2.B) and also the browsing swimlane (not shown Figure 2.B). However, the designer realizes that those three separate operations introduce code duplication in the functional design.

To eliminate this duplication the designer considers using the SOA design pattern “Service Façade” [1]. In the

service façade design pattern, the problem is that the tight coupling of the core service logic to its contracts can obstruct its evolution and negatively impact service consumers. As the *solution*, Façade logic is inserted into the service architecture to establish a layer of abstraction that can adapt to future changes to the service contract.

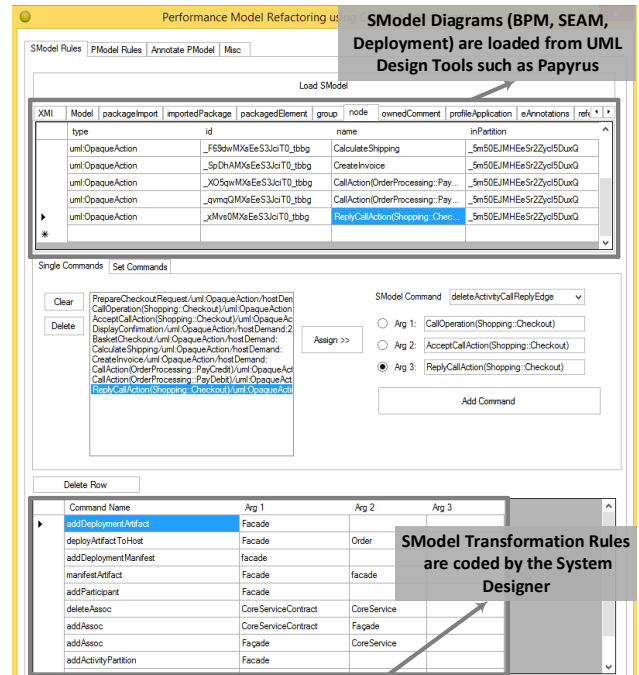


Figure 4 : Tool for Recording SModel Transformation Rules

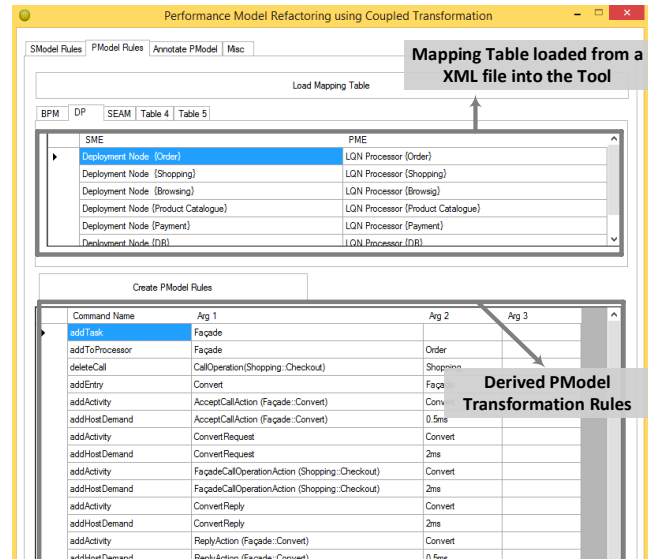


Figure 5 : Tool for automatic derivation of PModel Rules

Concerned that the façade overhead might impair the system performance, the designer applies the present technique. The designer first binds the pattern roles and records the necessary SModel changes (as in Section V, and the screenshot of Figure 4), using the base SOA design loaded from a standard UML modeling tool (e.g. Papyrus). The recorded rules and the Mapping Table are used by the

coupled transformation tool (as in Section VI and the screenshot of Figure 5) to derive the PModel transformation rules. The PModel transformation rules are applied to the LQN model shown in Figure 2.D, giving a performance model which is partly shown in Figure 6 below.

To illustrate how performance issues can be revealed, the performance was estimated for a range of user populations. For each N users in group “users1”, there were 2N in “users2”, and N/2 in “users3”. N ranged from 2 to 220, so the total users ranged from 7 to 770. Figure 7 shows the response times for the three groups of users and for both patterns. It shows that the groups have the same response time, and under heavy loads (which are also the conditions in which the system resources are efficiently utilized) the Façade pattern imposes about 30% additional delay in response time. This penalty is the price for the benefits it provides to the system architecture by preparing it for future changes to the service. An alternative view of the penalty is that it reduces the user population that a deployed system can serve with a given target response time.

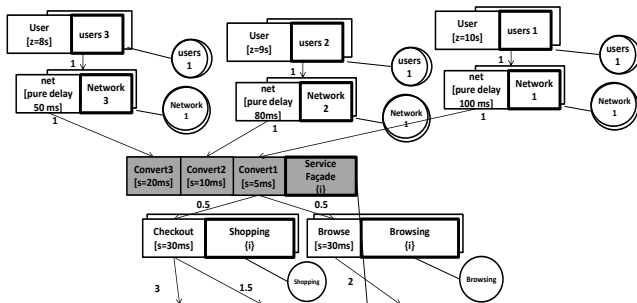


Figure 6: Partial Refactored PModel (Façade applied)

VIII. CONCLUSION

This paper describes a process and tools for interpreting a software pattern in terms of the corresponding change in a performance model of the software, to support an immediate analysis of the performance effects of using a pattern. It helps the system designer to choose a pattern that has acceptable performance impact, and to choose between alternatives. It provides the system designer with a systematic approach and tool for formally recording those changes for the SOA design and from these it automatically derives the performance model changes. Coupling the transformations ensures that the performance analysis remains in sync with the software changes, and relates the resource and performance changes back to the pattern.

The use of the process and tools was illustrated by an extensive example which applied the Façade pattern to a Browsing and Shopping system design, and by an analysis which compared its impact to that of the Concurrent Contracts pattern. The performance cost of the Façade pattern is a significant increase in response time under load, which could influence the development of the design.

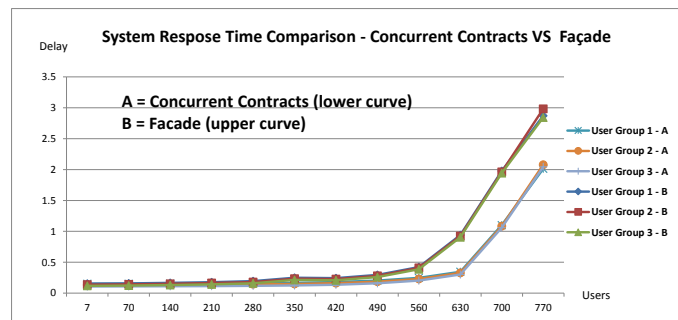


Figure 7: System Response Time (ms) for (A) Concurrent Contracts and (B) Façade patterns

ACKNOWLEDGMENT

This work was supported by the Ontario Centers of Excellence and by the Natural Sciences and Engineering Research Council of Canada (NSERC) through its Discovery Grant program.

REFERENCES

- [1] T. Erl, *SOA Design Patterns* Boston, MA: Prentice Hall PTR, 2009.
- [2] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer, "Performance by Unified Model Analysis (PUMA)," *WOSP '05 Proceedings of the 5th international workshop on Software and performance*, Palma de Mallorca, Illes Balears, Spain, 2005, pp. 1 - 12.
- [3] N. Mani, D. Petriu, and M. Woodside, "Propagation of Incremental Changes to Performance Model due to SOA Design Pattern Application," *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13)*, Research Papers Track Prague, Czech Republic, 2013, pp. 89-100.
- [4] B. Elvesater, C. Carrez, P. Mohagheghi, A. Berre, S. G. Johnsen, and A. Solberg, "Model-driven Service Engineering with SoaML," in *Service Engineering*, S. Dustdar and F. Li, Eds., Springer, 2011, pp. 25-54.
- [5] Object Management Group, "A UML Profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems)," Version 1.1, formal/2011-06-02.
- [6] R. B. France, D.-K. Kim, S. Ghosh, and E. Song, "A UML-Based Pattern Specification Technique," *IEEE Trans. Software Eng.*, vol. 30, pp. 193-206, 2004.
- [7] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi, "Enhanced Modeling and Solution of Layered Queueing Networks," *IEEE Trans. on Software Eng.*, vol. 35, pp. 148-161, 2009.
- [8] P. Haas, *Stochastic Petri Nets: Modelling, Stability, Simulation* Springer-Verlag, New York, 2002.
- [9] M. Alhaj and D. Petriu, "Traceability Links in Model Transformations between Software and Performance Models," in *SDI 2013: Model-Driven Dependability Engineering*, vol. 7916, F. Khendek, M. Toeroe, A. Gherbi, and R. Reed, Eds., Springer, 2013, pp. 203-221.
- [10] C. U. Smith and L. G. Williams, *Performance Solutions : A Practical Guide to Creating Responsive, Scalable Software*. Boston, MA: Addison Wesley, 2002.
- [11] V. Cortellesa, A. D. Marco, R. Eramo, A. Pierantonio, and C. Trubiani, "Digging into UML models to remove performance antipatterns," *Proceeding of ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems*, Cape Town, 2010, pp. 9-16.
- [12] J. Xu, "Rule-based automatic software performance diagnosis and improvement," *Proceeding of 7th Intl Workshop on Software and Performance*, Princeton, NJ, USA, 2008, pp. 1-12.
- [13] Object Management Group, "Query/View/Transformation (QVT) " Version 1.2 , formal/2015-02-01.

On the Specification of Model Transformations through a Platform Independent Approach

Magalhães, A.P.
Exact and Earth Science Department
State University of Bahia
Salvador, Brazil
anapatriciamagalhaes@gmail.com

Andrade, A.; Maciel, R.S.P.
Science Computer Department
Federal University of Bahia
Salvador, Brazil
{aline,ritasuzana}@dcc.ufba.br

Abstract— Transformations are key artifacts in the MDD (Model Driven Development) approach: a software development project can be defined through a transformation chain converting source models into target models until code, enabling development process automation. Transformations can be complex and demand software processes, languages and techniques to improve their development in order to increase reuse, portability, correctness, and so on. In this context we propose a framework to develop model transformations using MDD. This paper presents a Model Transformation Profile (MTP) defined as the domain specific language of the framework.

Keywords-Transformation profile, transformation specification, transformation metamodel.

I. INTRODUCTION

Model Driven Development (MDD) [7] is a paradigm that makes intensive use of models to represent systems at different level of abstraction (specification, design and code). A key element of the MDD approach is the transformation chain which is responsible for the conversion of source into target models until code generation. Transformations play an important role in MDD because they enable the automation of the model generation process, encapsulating knowledge and strategies used in the development of the software.

Despite the importance of models for the MDD approach, transformations are usually specified in an ad-hoc way using natural language and are implemented directly in code [4]. This practice leads to poor documentation which hampers the evolution of the transformation and makes it difficult to use software engineering good practices such as design patterns and reuse. In order to change this scenario some works have been proposed [2][3][4] to cover specific aspects of transformation development (e.g. transformation design).

In this context, we propose a MDD framework for model transformation development that comprises: (i) a MDD transformation development process, which guides developers through activities to produce transformation software; (ii) a profile, named Model Transformation Profile (MTP), to support the modeling process activities; and (iii) a tool to partially automate the modeling and transformations tasks of the process. In this paper we present the MTP profile whose first ideas were outlined in [6]. The MTP profile presented here has been improved from that incorporating another abstraction level, MTP_{LowDesign}, for the specification of transformation

behavior. New concepts and attributes have also been added in the other levels and we have developed a validation using experimental software engineering techniques to measure the quality of the profile.

MTP provides concepts to specify model transformations from requirements to design independent of platform. The produced transformation models can be transformed in a specific platform and then in code in different transformation languages (e.g. QTV [9], ATL [1]), increasing productivity and portability. MTP raises the abstraction level of the transformation development from code to model in a platform-independent way, abstracting some implementation details of specific transformation languages.

The rest of this paper is organized as follows: section 2 discusses the current development approaches to model transformation; section 3 briefly introduces our MDD framework; section 4 describes the MTP profile giving some examples; section 5 presents some results from a MTP validation; and finally Section 6 presents our conclusions.

II. RELATED WORKS

Our framework uses a visual UML profile as a modeling language, so we focus on comparing our proposal with existing visual approaches. Furthermore, we attempt to analyze the coverage of these works concerning the phases of a transformation development life cycle.

In MOF Query/View/Transformation (QVT) [9] a model transformation can be represented diagrammatically in two ways: using the UML class diagram or using a transformation diagram. The complexity of the QVT metamodels makes the diagram verbose and difficult to understand and the transformation diagram brings new notation with no portability to UML tools.

There are some works that focus on specific aspects of the transformation development. In [3] the authors propose a visual, formal, declarative specification language (graph based) focusing on transformation correctness, but it does not deal with implementation as we do. The work [14] focuses on internal composition of transformations. It generalizes composition mechanisms for rule-based transformation languages in order to provide executable semantic to them. Our proposal, on the other hand, works with external composition. In [15] generic programming is used to define reusable model

transformations. We follow another direction through a UML profile to support the development of model transformation models independent of platform such that models are reused in transformations in different languages.

The works [4][2] are more closely related to the one presented in this paper. TransML [4] proposes a family of languages with diagrams for the entire development life cycle providing support for specification, analysis, design and code. However, the proposed diagrams use a UML heavy extension and new notations that make it difficult to integrate with the existing UML tools which are usually adopted. MeTaGen [2] proposes metamodels for transformation design and tools generate code automatically or semi automatically. The main difference between this work and ours is that it focuses on design, not considering the requirement specification level and it uses textual language for transformation specification whereas we use a profile that is a visual language.

In summary, although existing works agree that transformation development requires a software life cycle, they usually focus on an individual phase of development lacking an entire process to transform the transformation model in code. We propose an integrated framework with a visual modeling language specialized from the UML standard that covers transformation development from requirements to code.

III. MDD TRANSFORMATION FRAMEWORK

The main goal of MDD Framework is to provide a process to develop model transformations suitable for a transformation domain, covering the entire software development life cycle integrated into a standard modeling language. Fig. 1 shows its main elements: (i) the MDD Transformation Development Process; (ii) the Model Transformation Profile (MTP); and (iii) a tool to (partially) automate the process.

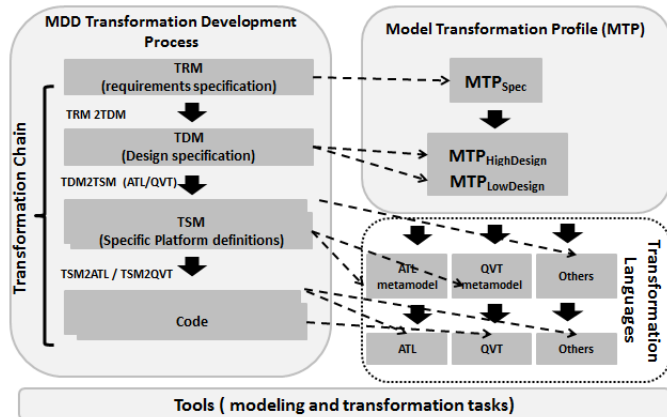


Figure 1. MDD Transformation Framework overview

The MDD Transformation Development Process aims to guide developers step by step on the development of model transformations. The process is specified according to SPEM [8] metamodel and comprises tasks that lead from requirements specification until code. Specification starts modeling the TRM (*Transformation Requirements Model*) which comprises requirements and analysis tasks. From requirements a semi-automatic transformation generates the first release of the TDM (*Transformation Design Model*) which aims to model the

design and architecture of the transformation software. Tasks include the definition of *what* might be transformed in what (high design), transformation structure (architecture) and *how* transformation should be performed (low design). This specification is then transformed into TSM (*Transformation Specific Model*) which refers to specific languages to then generate code. We provide generation for TSM in ATL language, due to its wide use in MDD projects to develop transformations, or QVT language, the OMG standard to design model transformations. The MTP Profile is defined to support the modeling tasks of the proposed process. It is detailed in the following sections.

IV. MODEL TRANSFORMATION PROFILE (MTP)

The MTP Profile is a modeling language that extends UML for the model transformation domain. Its main goal is to provide a platform-independent visual language, suitable for a model transformation domain which can be used to develop model transformations at a high abstraction level (TRM and TDM models). The profile covers the definition of model-to-model unidirectional transformation using a visual language.

In order to specify the MTP we define: an abstract syntax, represented by metamodels, with the concepts of the transformation domain; a static semantic, described with a set of OCL constraints which determine the well-formed criteria of the instantiated models; and a set of stereotypes and their UML specialized metaclasses. MTP is divided into three parts, MTP_{Spec} , $MTP_{HighDesign}$ and $MTP_{LowDesign}$.

The main goal of the MTP_{Spec} is to provide definitions for the specification and analysis of transformation requirements. Its abstract syntax is shown in Fig. 2.

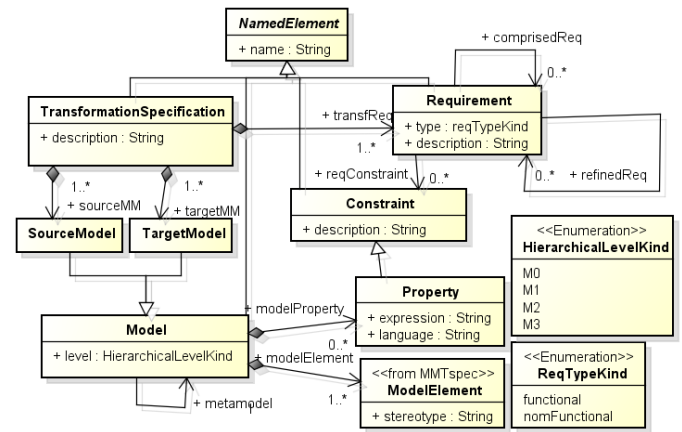


Figure 2. MTP_{Spec} Metamodel

At specification level a *TransformationSpecification* has a *name*, a *description* and is composed of a *Requirement*. *Requirement* may be refined in other requirements (*refinedReq* association) and may also be composed of other requirements (*comprisedReq* association). *Constraint* can be specified for requirements in natural language. A *Requirement* has a *name*, a *description* and a *type* that identifies if it is functional or non-functional. *TransformationSpecification* is also composed of source (*sourceMM*) and target (*targetMM*) metamodels. Models, metamodels and metametamodels are represented by the concept *Model* and have a *level*. This *level* indicates the

OMG model layer in which they are defined (e.g. M3). Properties of specific domains can be specified in *Property*.

The concrete syntax of the *MTP* consists on a package of stereotypes associated to UML metaclasses. For example, the *TransformationSpecification* *MTP* concept is specialized as an *actor* in UML. Due to lack of space only part of the concrete syntax of *MTP_{spec}* is shown in Tab.1.

MTP_{Spec} supports the Transformation Process enabling requirements elicitation and analysis in Use Case and Classes diagrams.

TABLE I. PART OF *MTP_{spec}* STEREOTYPE AND METACLASSES

Stereotype	Metaclass
<< Transformation Specification >>	Actor, class
<<Requirement>>	Use case, class
<<Model>>	Class, attribute, package

MTP also comprises a set of OCL constraints with additional well-formed criteria used on model instantiation. Due to lack of space they are not presented here.

MTP_{Design} provides the necessary definitions for the design and architecture specification of the transformation. The profile was organized in two packages, named *MTP_{HighDesign}* and *MTP_{LowDesign}*.

MTP_{HighDesign} defines *what* will be transformed in what. Its abstract syntax is presented in Fig. 3.

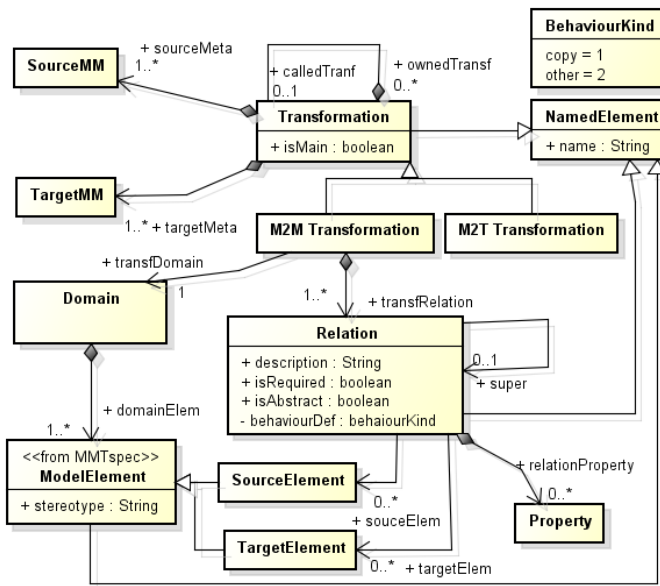


Figure 3. *MTP_{HighDesign}* metamodel

A *Transformation* may be composed of other transformations, enabling reuse. *Transformation* is specialized in *M2M Transformation*, to represent model-to-model transformations and *M2T Transformation*, to represent model-to-text transformations (not detailed in this work). *M2M Transformation* defines a *Domain* and it is composed of *Relation*. A *Domain* specifies which *Element* of the source/target metamodel will be considered by the

transformation. It will be used to verify transformation completeness (section 4A). A *Relation* has a *name*, a *description* to document it and might be concrete or abstract (attribute *isAbstract*) allowing *Relation* inheritance. The attribute *isRequired* indicates if it is automatically processed when transformation is executed or if it is explicitly invoked by another *Relation*. A *Relation* may also have a set of *Property* (e.g. OCL constraints). The main purpose of a *Relation* is the definition of relationships between elements from source to target metamodels (*SourceElement* and *TargetElement*). It is possible to define many kinds of relationships: zero-to-one; zero-to-many; one-to-one; one-to-many; many-to-many; one-to-zero and many-to-zero as shown by the multiplicity of the *sourceElem* and *targetElem* association.

MTP_{HighDesign} supports the TDM (Transformation Design Model) specification through the use of classes and component diagrams. Class diagrams are used for the specification of *Relation* between elements from source to target metamodels in order to hierarchically organize the rules of a transformation, providing transformation inheritance. Component diagrams are used to model transformation chains: each transformation in a chain is represented by a component whose interfaces specify the source and target models and metamodels.

The *MTP_{LowDesign}* defines *how* *Relation* converts elements from the source model into elements of the target model. Fig. 4 shows the *MTP_{LowDesign}* metamodel. The *Relation* concept (from *MTP_{HighDesign}*) is now detailed by the *Rule* concept which is composed of *SourceElementRule* and *TargetElementRule*. For each *SourceElement* of *Relation* a *SourceElementRule* is modeled for the corresponding *Rule* and a reference (*ref* attribute) must be defined. This reference will be later used in expression definitions. A *SourceElementRule* may be associated to *Condition* (defined in the *exp* attribute) that must be satisfied for the rule to be executed. *TargetElementRule* comprises a set of *Configuration* that defines how the Attributes of the *TargetElementRule* will be initialized when generated. The *Configuration* is specified through the definition of an expression (*exp* attribute) that will be assigned to attributes of the associated *TargetElementRule*. Expressions are defined using a textual language. *MTP_{LowDesign}* supports transformation process through the use of class diagrams.

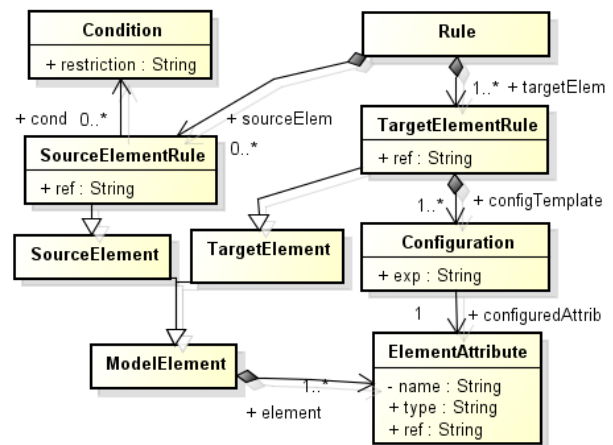


Figure 4. *MTP_{LowDesign}* metamodel

A. MTP and Transformation Properties

There are some properties that assure transformation quality, such as syntactic and semantic correctness and completeness [10][11].

The syntax correctness defines the conformity between models and metamodels and the semantic correctness consists of property preservation from source to target models. We define some OCL constraints in order to guarantee conformance of model transformation models which are instances of MTP. Our framework foresees the specification of semantic properties through the *Property* concept (Fig.2). Therefore it is possible to specify a set of properties in the early stages of the transformation definition and this set can be extended with other properties at the application level.

A transformation is complete if and only if for each element of the source metamodel there is a corresponding element in the target metamodel mapped by the transformation. In order to address completeness, MTP provides the *Domain* concept (Fig.3) which identifies the set of elements of source/target metamodels that are mapped by the transformation. Based on the *Domain* definition and on OCL constraints, completeness can be verified after the instantiation of the model transformation model.

V. MTP VALIDATION

The validation consists in the assessment of the expressiveness of MTP profile constructors. To assist validation we followed the guidelines for software engineering experimentation presented in [13] and use GQM [12] to summarize our goal (Fig.5). The questions underlying the validation are: Q1: Are the MTP constructors sufficient to specify transformations written in ATL/QVT? Q2: Is it necessary to add new constructors in MTP to enable the transformations specification written in ATL/QVT? Q3: Are the selected UML diagrams sufficient to specify transformations?

<p>Analyze the MTP profile constructors For the purpose of evaluating expressiveness With respect to coverage of the profile constructors and specification completeness From the perspective of transformation developers In the context of existing transformations developed in ATL/QVT languages</p>
--

Figure 5. Experiment goal according to QGM template

We use several measures as dependent variables such as the amount of used constructors, the need of new constructors, the amount of changes on existing constructors, the level of specification detail and the used UML diagrams.

The validation process lasted five months and was divided into two stages: an *initial test* and the *main validation*. These two stages were performed by our research group in laboratory and consisted of using MTP to specify transformations already developed in ATL / QVT languages.

According to validation, related to questions Q1 and Q2, we concluded that MTP constructors are sufficient to specify transformations without the necessity to add new constructors. Related to question Q3 we observed that, after including component diagram in the *initial test*, the selected UML

diagrams were sufficient to specify the transformations. Therefore, we considered that the MTP was stable enough to be used on the framework case study.

VI. CONCLUSIONS AND FUTURE WORKS

The Model Transformation Profile presented in this paper is a modeling language that is part of a framework to develop model transformation using MDD.

MTP represents transformations concepts at different abstraction levels, covering many phases of transformation development such as requirements, analyses and design enabling transformation modeling independent of platform. In this sense it postpones specific platform definitions to later phases of development. As a UML profile MTP takes advantage of the wide use of UML in both industry and academy benefiting from tools already used by the development community. The validation of the profile demonstrated that MTP concepts cover most transformation specification needs and that UML diagrams were suitable for transformation specifications. Therefore, we consider MTP to be stable for use in real projects.

We are currently specifying a MTP behavioral semantics in order to enable simulation of transformation specification.

REFERENCES

- [1] ATL Project - <http://www.eclipse.org/m2m/atl/>
- [2] Bollati, V., Vara, J., Jiménez, A., Marcos, E. "Applying MDE to the (semi-)automatic development of model transformations." Information and Software Technology, pp.699-718, Elsevier, 2013.
- [3] Guerra, E.; Lara, J.; Kolovos, D.; Paige, R. "A Visual Specification Language for Model-to-Model Transformations." IEEE Symposium on Visual Languages and Human-Centric Computing, DOI 10.119/VLHCC.2010.25, 2010.
- [4] Guerra, E.; Lara, J.; Kolovos, D.; Paige, R.; Santos O. "TransML: A Family of Languages to Model Model Transformations." Models, 2010, DOI 10.1007/s10270-011-0211-2, Springer Verlag, . 2010.
- [5] Iacob, M., Steen, M., Heerink, L.: "Reusable Model Transformation " Pattern. In 3M4EC'08, pages 1-10, 2008.
- [6] Magalhães, A. P., Andrade, A. ; Maciel, R.S.P. "MTP: Model Transformation Profile." In: SBCARS, 2013, Brasilia. p. 109-118 2013.
- [7] Mellor,S.; Clark, A.; Futagami, T. "Model Driven Development" IEEE Software,2003
- [8] OMG. Software Process Engineering Metamodel Specification, Version 2.0, (formal/08-04-01).2008.
- [9] QVT specification - <http://www.omg.org/spec/QVT/1.0/PDF/>
- [10] Lano, K.; Clark, D. "Model Transformation Specification and Verification." The 18th International Conference on Quality Software, IEEE, 2008.
- [11] Mens, T.; Gorp, P.V. "A Taxonomy of Model Transformation." Elsevier Electronic Notes in Theoretical Computer Science 152 pp. 125-142 2006.
- [12] Solingen, R. Basili, V.;Caldiera,G.; Rombach, H.D. Goal Question Metric (GQM) Approach. John Wiley & Sons. Inc., 2002.
- [13] Wohlin, C. Aurum, A. Towards a decision-making structure for selecting a research design in empirical software Engineering. Empir Software Eng DOI 10.1007/s 10664-014-9319-7. Springer, 2014.
- [14] Wagelaar, D.; Tisi, M.; Cabot, J.; Jouault, F. Towards a General Composition Semantics for Rule-Based Model Transformation. MODELS, 2011.
- [15] Cuadrado, J.; Guerra, E.; Lara, J. Generic Model Transformations: Write Once, Reuse Everywhere. ICMT, 2011.

Improved Metrics for Non-Classic Test Prioritization Problems

Ziyuan Wang^{1,2} Lin Chen²

¹ School of Computer, Nanjing University of Posts and Telecommunications, Nanjing, 210003

² State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023

Email: wangziyuan@njupt.edu.cn

Abstract—The average percent of faults detected (*APFD*) and its variant versions are widely used to evaluate prioritized test suite’s efficiency. However, *APFD* is only available for classic test case prioritization, where all prioritized test suites under comparison contain the same number of test cases. If people overlook this phenomenon, they may obtain incorrect results in some non-classic scenarios, where prioritized test suites have different sizes. In addition, it can’t precisely illustrate the process of fault detection. Besides the *APFD*, most of its variants have similar problems. This paper points out these limitations in detail, by analyzing the physical explanation of *APFD* series metrics. To avoid limitations, a series of metrics including *RAPFD*, *RAPFD_C*, *RAPFD_W* and *RAPFD_{CW}* are proposed for different types of scenarios. All proposed metrics refer to both the speed of fault detection and the constraint of testing resource. There is an example in this paper showing that proposed metrics provide much more precise illustrations of fault detection process and fault detection efficiency of test suite.

Keywords—software testing, test case prioritization, fault detection efficiency, metric

I. INTRODUCTION

There is usually a contradiction in test case evolution. For the purpose of rapid version release, we usually need a speedup regression testing to save test resource (e.g. consumed time). However, for the purpose of higher-quality, we want to run test cases as many as possible. The contradiction between them tells people it is necessary to apply some test case optimization techniques to increase the effectiveness and efficiency of regression testing. As one of test case optimization techniques, the test case prioritization technique has been widely used to improve the efficiency of software testing.

The test case prioritization technique aims to schedule test cases in an order, to form a prioritized test suites. The classic test case prioritization problem is defined as follows. Giving an initial test suite T_{init} , the test case prioritization technique aims to find a prioritized test suite $\sigma \in PT$ such that:

$$(\forall \sigma')(\sigma' \in PT)(\sigma \neq \sigma')[f(\sigma) > f(\sigma')] \quad (1)$$

Where PT is the set of all permutations of T_{init} (PT collects all possible prioritized test suites that contain all test cases in T_{init}). And f is an objective function from PT to real number as the award value of prioritized test suite [1].

There could be many possible objective functions for test case prioritization problem. People usually restrict attention to the speed of detecting faults. Therefore, an objective function called *average percent of faults detected* (*APFD*) is proposed as a metric to evaluate the speed of detecting faults [1]. It helps

people to compare which prioritized test suite $\sigma \in PT$ detects faults more rapidly. There are also some variants of *APFD*, including the *normalized average percent of faults detected* (*NAPFD*) [2], the *cost-cognizant weighted average percent of faults detected* (*APFD_C*) [3], and etc. In this paper, we use the term *APFD series metrics* to jointly call them.

APFD series metrics are designed for the classic test case prioritization problem that defined in Equation 1, which implies an assumption that all prioritized test suites in PT must contain all test cases in the initial test suite (for each $\sigma \in PT$, $|\sigma| = |T_{init}|$). However, besides the classic problem, there may be some other non-classic test case prioritization problems, where above assumption does not hold:

- (1) *Time-aware test case prioritization* selects and prioritizes test cases under the time constraint [6]. Different selection algorithms may produce prioritized test suites with different sizes.
- (2) *Test goal prioritization* schedules test goals and generates test cases for important test goals earlier [7]. It may leads to different prioritized test suites because of the different orders of test goals.
- (3) *Test case re-generation prioritization strategy* incorporates test prioritization into test generation (e.g. in combinatorial testing [5]). Different algorithms may generate prioritized test suites with different sizes.
- (4) *Test case reduction* [8] and test case prioritization are often incorporated in test case optimization. Different test case reduction algorithms may output test suites with different sizes.

In these novel scenarios, heterogeneous candidate prioritized test suites may contain only partial test cases in initial test suite (e.g. time-aware test case prioritization and test case reduction), or sometimes be not concerned with any initial test suite at all (e.g. test case re-generation prioritization and test goal prioritization). *APFD* and existing variants can hardly work to evaluate and compare these candidate prioritized test suites, since the number of test cases that contained in each candidate prioritized test suites may be varying.

And besides the limitation about test suites’ sizes, there is another limitation of existing *APFD* series metrics. That is they cannot precisely illustrate the process of fault detection in real world.

Therefore, we need some improved metrics for non-classic test case prioritization. In this paper, we propose an improved metric *Relative-APFD* (*RAPFD* for short), which relates to a given testing resource constraint (determine how many test cases could be run), to replace *APFD* and *NAPFD*. And

furthermore, we discuss the test costs and fault severities, and propose the metric *Relative-APFD_{CW}* (*RAPFD_{CW}* for short) to replace *APFD_C*. Examples show us that proposed metrics could provide much more precise illustrations of fault detection process and fault detection efficiency of test suite.

II. EXISTING APFD SERIES METRICS

Using notations that introduced in the ref. [6], we briefly introduce existing *APFD* series metrics in this section.

1) *APFD*: Let σ be a prioritized test suite under evaluation, Φ the set of fault contained in the software, and $TF(\phi, \sigma)$ the index of the first test case in σ that exposes fault $\phi \in \Phi$. The *APFD* of σ is [1]:

$$APFD(\sigma) = 1 - \frac{\sum_{\phi \in \Phi} TF(\phi, \sigma)}{|\sigma||\Phi|} + \frac{1}{2|\sigma|} \quad (2)$$

2) *NAPFD*: Sometimes, there may be non-detected fault that can't be detected by any test cases in σ . For each non-detected fault $\phi \in \Phi$, Walcott et al. set $TF(\phi, \sigma) = |\sigma| + 1$ as a penalty that may make *APFD* value to become negative [6]. To avoid the problem of negative award value, Cohen et al. set $TF(\phi, \sigma) = 0$ and define an improved *NAPFD* [2]:

$$NAPFD(\sigma) = p - \frac{\sum_{\phi \in \Phi} TF(\phi, \sigma)}{|\sigma||\Phi|} + \frac{p}{2|\sigma|} \quad (3)$$

Where p is the rate of faults detected by σ :

$$p = \frac{|\{\phi \in \Phi | TF(\phi, \sigma) \neq 0\}|}{|\Phi|}$$

3) *APFD_C*: Another improvement for *APFD* is to take the test costs and the fault severities into consideration. Let C_i be the cost of the i -th test case in σ ($i = 1, 2, \dots, |\sigma|$), S_ϕ the severity of the fault $\phi \in \Phi$. For the scenario where there is not any non-detected faults, the *APFD_C* is [3]:

$$APFD_C(\sigma) = \frac{\sum_{\phi \in \Phi} (S_\phi \times (\sum_{i=TF(\phi, \sigma)}^{|\sigma|} C_i - \frac{1}{2} C_{TF(\phi, \sigma)}))}{\sum_{i=1}^{|\sigma|} C_i \times \sum_{\phi \in \Phi} S_\phi} \quad (4)$$

There is special case of *APFD_C*, called *APFD_{TA}*, for the scenario where fault severities are uniform [4].

4) *NAPFD_C*: Similar to the *APFD*, the original version of *APFD_C* can not handle non-detected faults, since there is not any definition of $TF(\phi, \sigma)$ for non-detected faults.

Here we propose the *normalized cost-cognizant weighted average percent of faults detected* (*NAPFD_C*), by defining $TF(\phi, \sigma) = 0$ for each non-detected fault $\phi \in \Phi$ and setting $C_0 = 0$ for the dummy test case with index 0:

$$NAPFD_C(\sigma) = p_c - \frac{\sum_{\phi \in \Phi} (S_\phi \times \sum_{i=1}^{TF(\phi, \sigma)} C_i)}{\sum_{i=1}^{|\sigma|} C_i \times \sum_{\phi \in \Phi} S_\phi} + \frac{\sum_{TF(\phi, \sigma) \neq 0} (S_\phi \times C_{TF(\phi, \sigma)})}{2 \times \sum_{i=1}^{|\sigma|} C_i \times \sum_{\phi \in \Phi} S_\phi} \quad (5)$$

Where p_c is the rate of total severities of faults detected by σ :

$$p_c = \frac{\sum_{TF(\phi, \sigma) \neq 0} S_\phi}{\sum_{\phi \in \Phi} S_\phi}$$

It is evident that *NAPFD_C* will be equivalent to *APFD_C* in the scenarios where there is not any non-detected fault. And *NAPFD_C* will be equivalent to *NAPFD* in the scenarios where both test costs and fault severities are uniform.

III. PHYSICAL EXPLANATIONS OF EXISTING METRICS

For a test suite with $|\sigma|$ test cases, we can use $|\sigma| + 1$ discrete points to illustrate the relationship between the percent of faults detected (y-axis) and the percent of test cases run (x-axis). If we connect all these points by a curve, *APFD* and *NAPFD* show the area under the curve. E.g., we select 5 test cases from Table 1 to form a prioritized test suite $\sigma_1 : T_3 - T_5 - T_2 - T_4 - T_1$. It detects 3 faults using 1 test case, 5 faults using 2 test cases, 7 faults using 4 test cases, and all 8 faults using all 5 test cases. So $APFD(\sigma_1) = NAPFD(\sigma_1) = 0.6$ by drawing the curve that connects 6 discrete points $(0, 0)$, $(\frac{1}{5}, \frac{3}{8})$, $(\frac{2}{5}, \frac{5}{8})$, $(\frac{3}{5}, \frac{5}{8})$, $(\frac{4}{5}, \frac{7}{8})$, and $(1, 1)$ (see Fig. 1).

Table 1. Faults Detected by Test Cases

	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_8
T_1							x	x
T_2		x						
T_3		x		x			x	
T_4	x		x					
T_5				x	x	x		
T_6	x		x					x

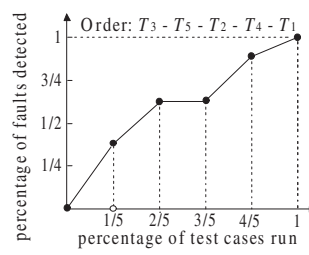


Fig. 1. *APFD*(σ_1)

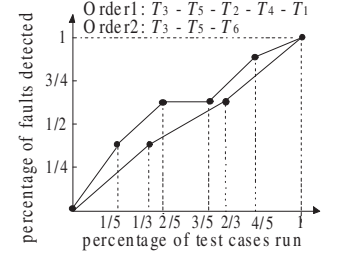


Fig. 2. Compare σ_1 and σ_2

The *APFD_C* and *NAPFD_C* have the similar physical explanations, if we replace the number of detected faults with the total severities of detected faults, and replace the number of test cases run with the costs that consumed by test cases.

Besides the above physical explanations, *APFD* series metrics can help people to control the risk of exceptional termination in testing process under the constraint of testing resource. As well known, during the software evolution, resource (including time) that distributed to software testing are often limited since the delay of develop, the deadline of release, and etc. So there is usually not enough time to run all test cases.

Theorem 1. Let C_i be the consumed time of the i -th test case $T_i \in \sigma$ ($i = 1, 2, \dots, |\sigma|$), and S_ϕ the severity of fault $\phi \in \Phi$. Suppose the variable $t \in [0, \sum_{i=1}^{|\sigma|} C_i]$ be the time

of the moment testing process terminates exceptionally. The $NAPFD_C(\sigma)$ is the mathematical expectation (or expected value) of the percent of total severities of faults detected before the termination, if following two assumptions hold:

- 1) $t \sim U[0, \sum_{i=1}^{|\sigma|} C_i]$: That is t follows the continuous uniform distribution with parameters $(0, \sum_{i=1}^{|\sigma|} C_i)$.
- 2) During the running of the i -th test case $T_i \in \sigma$, the total severities of its newly detected faults grow linearly with its consumed time.

Proof is omitted since the length constraint.

According to this theorem, testers can use $NAPFD_C$ to select the prioritized test suite that make more contributions under the constraint of testing resource. As we claimed previously, $APFD_C$, $NAPFD$ and $APFD$ are all the special cases of $NAPFD_C$, so they have the similar properties in scenarios where (1) $p_c = 1$, (2) costs / severities are uniform, (3) $p = 1$ and costs / severities are uniform, respectively.

IV. LIMITATIONS OF EXISTING METRICS

There are some limitations of $APFD$ series metrics though they have practical explanations.

A. Test Suites' Sizes or Total Costs

$APFD$ series metrics are designed for the classic test case prioritization problem that defined in Equation 1, where all prioritized test suites under evaluation contain the same number of test cases. However, there may be some other types of scenarios in software evolution and testing evolution, including time-aware test case prioritization, test goal prioritization, test case re-generation prioritization, test case reduction, and etc, where people need to compare prioritized test suites in which the numbers of contained test cases are different. So there is a question: can $APFD$ series metrics work in these scenarios?

There is a condition in Theorem 1 that $t \sim U(0, \sum_{i=1}^{|\sigma|} C_i)$. Considering two prioritized test suites σ_1 and σ_2 where $\sum_{i=1}^{|\sigma_1|} C_i \neq \sum_{i=1}^{|\sigma_2|} C_i$, there are $t_{\sigma_1} \sim U(0, \sum_{i=1}^{|\sigma_1|} C_i)$ and $t_{\sigma_2} \sim U(0, \sum_{i=1}^{|\sigma_2|} C_i)$ respectively. Though both t_{σ_1} and t_{σ_2} follow continuous uniform distribution, their parameters are different: the former follows the distribution with parameters $(0, \sum_{i=1}^{|\sigma_1|} C_i)$ while the latter follows the distributions with parameters $(0, \sum_{i=1}^{|\sigma_2|} C_i)$. It is unfair, and even meaningless, to compare mathematical expectation by using $NAPFD_C$, when probability density functions are different. Therefore, $APFD$ series metric are not suitable to be used to compare prioritized test suites with different sizes (for $APFD$ and $NAPFD$) or different total costs (for $APFD_C$ and $NAPFD_C$).

Here we can take the test cases and faults in Table 1 as examples to show some incorrect results when use $APFD$ and $NAPFD$. The incorrect results could be extended to illustrate the limitations of $APFD_C$ and $NAPFD_C$, since $APFD$ and $NAPFD$ are special cases of them respectively.

- 1) For situation that all faults are detected by prioritized test suites, construct two prioritized test suite $\sigma_1 : T_3 - T_5 - T_2 - T_4 - T_1$ and $\sigma_2 : T_3 - T_5 - T_6$. Note that both σ_1 and σ_2 detect all faults. Then we compare their $APFD$ values (also see Fig. 2):

$$\begin{aligned} APFD(\sigma_1) &= APFD_C(\sigma_1) = \frac{3}{5} \\ APFD(\sigma_2) &= APFD_C(\sigma_2) = \frac{1}{2} \end{aligned}$$

But, it is incorrect to say σ_1 is more efficient than σ_2 . After run 1 (or 2) test case(s), both σ_1 and σ_2 detect 3 (or 5) faults; after run 3 test cases, σ_2 detects all 8 faults while σ_1 detects only 5; and finally σ_1 need 2 more test cases to detect all 8 faults. It is clear that σ_2 detects faults more rapidly than σ_1 .

- 2) For situation that there are non-detected faults, construct two prioritized test suite $\sigma_3 : T_3 - T_2 - T_5$ and $\sigma_4 : T_3 - T_5$. Note that both σ_3 and σ_4 detect 5 faults. Then we get:

$$\begin{aligned} NAPFD(\sigma_3) &= \frac{17}{48} \\ NAPFD(\sigma_4) &= \frac{11}{32} \end{aligned}$$

But, it is incorrect to say σ_3 is more efficient than σ_4 . After run 1 test case, both σ_3 and σ_4 detect 3 faults; after run 2 test cases, σ_4 detects 5 faults while σ_3 detects 3 faults; and finally σ_3 need one more test case to detect 5 faults. It is clear that σ_4 detects faults more rapidly than σ_3 .

This phenomenon is often overlooked. There may be some incorrect and confused experimental results in the applications of $APFD$ series metrics in some previous papers. E.g. in ref. [6] and [2], authors used $APFD$ and $NAPFD$ respectively to compare prioritized test suites, in which the numbers of contained test cases are different, without any pretreatment.

B. Process of Fault Detection

$APFD$ series metrics can't precisely illustrate the process of fault detection in real world.

Note that the second condition of Theorem 1 is that the total severities of detected faults grow linearly with consumed time. In detail, during the running of one given test case, the number of newly detected faults (for $APFD$ and $NAPFD$) or the total severities of newly detected faults (for $APFD_C$ and $NAPFD_C$) grow linearly. Taking the prioritized test suite $\sigma_1 : T_3 - T_5 - T_2 - T_4 - T_1$ as an example, in the scenario where both test costs and fault severities are ignored, there is a continue function, which from the number of test cases run to the percent of faults detected, reflecting the process that σ_1 detects faults (see the curve in Fig. 1).

But factually, if a test case is still running, it cannot detect any faults, since it is impossible to check whether this test case is passed or failed before the end of running. It means that the function, which from the number of test cases run (or consumed time) to the percent of faults detected (or percent of total severities of detected faults), should be a step function in order to reflect the process of detecting faults. Also taking $\sigma_1 : T_3 - T_5 - T_2 - T_4 - T_1$ as an example, the corresponding step function is shown in Fig. 3. And the difference between continue function and step function is shown in Fig. 4.

So in computing mathematical expectation that explained in Theorem 1 for a prioritized test suite, there will be a margin of error. The margin of error may be very severe especially when the number of test cases is small. And if there are numerous test cases in prioritized test suite, we may accept such a approximation since the windage is minor.

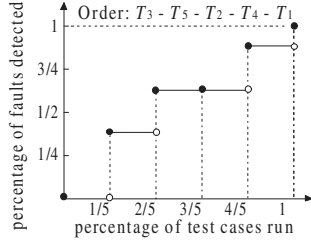


Fig. 3. Step Function of σ_1

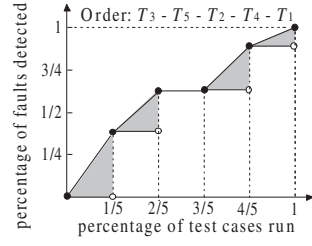


Fig. 4. Difference between Continue and Step Function

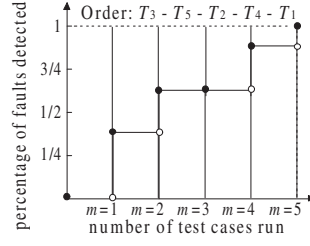


Fig. 5. $RAPFD(\sigma_1, m)$

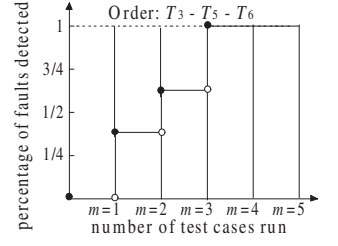


Fig. 6. $RAPFD(\sigma_2, m)$

V. IMPROVED METRICS

To avoid above limitations, we propose a series of improved metrics especially for non-classic test prioritization problems.

A. Relative-APFD

For two or more prioritized test suites that contain different number of test cases, if we want to evaluate and compare how rapidly they detect faults in the scenario where both test costs and fault severities are ignored, a fair testing resource should be provided firstly. Here the testing resource, which could be described as a positive integer m , is considered as a constraint:

- 1) $m < |\sigma|$: at most m test cases in σ will run.
- 2) $m \geq |\sigma|$: all the $|\sigma|$ test cases in σ will run.

By using the testing resource constraint, we propose an improved metric called *relative average percent of faults detected* (*Relative-APFD* or *RAPFD* for short). The value of *RAPFD* dose not only depend on the test suites under evaluation, but also relate to the given testing resource constraint. And further, this metric could handle non-detected faults.

Formally, let σ be a prioritized test suite under evaluation, Φ the set of fault contained in the software, and $TF(\phi, \sigma)$ the position of the first test case in σ that exposes fault $\phi \in \Phi$ ($TF(\phi, \sigma) = 0$ for non-detected fault). For a given testing resource constraint m , the *RAPFD* of σ is defined as:

$$RAPFD(\sigma, m) = p_m - \frac{\sum_{\phi \in \Phi} RTF(\phi, \sigma, m)}{m \times |\Phi|} \quad (6)$$

Where

$$RTF(\phi, \sigma, m) = \begin{cases} TF(\phi, \sigma) & : m \geq TF(\phi, \sigma) \\ 0 & : m < TF(\phi, \sigma) \end{cases}$$

And p_m is the ratio of the number of faults detected by first m test cases in σ to the number of faults in Φ :

$$p_m = \frac{|\{\phi \in \Phi | RTF(\phi, \sigma, m) \neq 0\}|}{|\Phi|}$$

Taking the prioritized test suite $\sigma_1 : T_3 - T_5 - T_2 - T_4 - T_1$ as an example, for a given testing resource constraint m , $RAPFD(\sigma_1, m)$ shows the area surrounded by the x-axis, y-axis, the line with $x = m$, and the curve that reflects the step function of fault detection process of σ_1 . (See Fig. 5).

B. Relative-APFD_{CW}

Considering the scenarios that test costs and fault severities are varying, the given uniform testing resource constraint should be scaled by a positive real number m_c , which indicates that the consumed testing resource should be less or equal to m_c . Then we can propose the metric called *relative cost-cognizant weighted average percent of faults detected* (*Relative-APFD_{CW}* or *RAPFD_{CW}* for short).

Formally, let σ be a prioritized test suite under evaluation, Φ the set of fault contained in the software, and $TF(\phi, \sigma)$ the position of the first test case in σ that exposes fault $\phi \in \Phi$ ($TF(\phi, \sigma) = 0$ for non-detected fault). And let C_i be the cost of the i -th test case ($i = 1, 2, \dots, |\sigma|$), S_ϕ the severity of the fault $\phi \in \Phi$. For a given testing resource constraint m_c , the *RAPFD_C* of σ is defined as:

$$RAPFD_{CW}(\sigma, m_c) = p_{m_cw} - \frac{\sum_{\phi \in \Phi} (S_\phi \times \sum_{i=1}^{RTF_C(\phi, \sigma, m_c)} C_i)}{m_c \times \sum_{\phi \in \Phi} S_\phi} \quad (7)$$

Where

$$RTF_C(\phi, \sigma, m_c) = \begin{cases} TF(\phi, \sigma) & : m_c \geq \sum_{i=1}^{TF(\phi, \sigma)} C_i \\ 0 & : m_c < \sum_{i=1}^{TF(\phi, \sigma)} C_i \end{cases}$$

And p_{m_cw} is the ratio of the total severities of faults detected by σ within the testing resource constraint, to the total severities of all faults in Φ :

$$p_{m_cw} = \frac{\sum_{RTF_C(\phi, \sigma, m_c) \neq 0} S_\phi}{\sum_{\phi \in \Phi} S_\phi}$$

It is clear that *RAPFD_{CW}* is equivalent to *RAPFD* when both test costs and fault severities are uniform.

And further, for the scenario where only test costs / only fault severities are taken into consideration, we can utilize

$$RAPFD_C(\sigma, m_c) = p_{m_c} - \frac{\sum_{\phi \in \Phi} \sum_{i=1}^{RTF_C(\phi, \sigma, m_c)} C_i}{m_c \times |\Phi|} \quad (8)$$

And

$$RAPFD_W(\sigma, m) = p_{mw} - \frac{\sum_{\phi \in \Phi} (S_\phi \times RTF(\phi, \sigma, m))}{m \times \sum_{\phi \in \Phi} S_\phi} \quad (9)$$

respectively. Where

$$p_{mc} = \frac{|\{\phi \in \Phi | RTFC(\phi, \sigma, m_c) \neq 0\}|}{|\Phi|}$$

$$p_{mw} = \frac{\sum_{RTF(\phi, \sigma, m) \neq 0} S_\phi}{\sum_{\phi \in \Phi} S_\phi}$$

They are both special cases of $RAPFD_{CW}$

C. Physical Explanation

We take the $RAPFD_{CW}$ as example to analyze the physical meaning of improved metrics. $RAPFD$, $RAPFD_C$ and $RAPFD_W$ will be omitted, since they could be considered as a special case of $RAPFD_{CW}$.

Theorem 2. Let C_i be the consumed time of the i -th test case $T_i \in \sigma$ ($i = 1, 2, \dots, |\sigma|$), S_ϕ the severity of fault $\phi \in \Phi$. Suppose the variable $t \in [0, m_c]$ be the time of the moment testing process terminates exceptionally. The $RAPFD_{CW}(\sigma, m_c)$ is equal to the mathematical expectation (or expected value) of the percent of total severities of faults detected before the termination, if following two assumptions hold:

- 1) $t \sim U[0, m_c]$. It means that t follows the continuous uniform distribution with parameters $(0, m_c)$.
- 2) During the running of the i -th test case $T_i \in \sigma$, the total severities of its newly detected faults keep still, until the execution of T_i is finished.

Proof is omitted since the length constraint.

The theorem means that proposed $RAPFD$, $RAPFD_C$, $RAPFD_W$, and $RAPFD_{CW}$ could help people to control the risk of testing process too.

VI. EXAMPLES

We still take the test cases and faults that shown in Table 1 as examples to illustrate the advantage of proposed $RAPFD$. Considering $\sigma_1 : T_3 - T_5 - T_2 - T_4 - T_1$ and $\sigma_2 : T_3 - T_5 - T_6$. For testing resource constraint $m = 1, 2, 3, 4$, and 5 respectively, we compute $RAPFD$ for σ_1 and σ_2 (also see the area under the step functions in Fig. 5 and Fig. 6):

- 1) $RAPFD(\sigma_1, 1) = RAPFD(\sigma_2, 1) = 0$
- 2) $RAPFD(\sigma_1, 2) = RAPFD(\sigma_2, 2) = \frac{3}{16}$
- 3) $RAPFD(\sigma_1, 3) = RAPFD(\sigma_2, 3) = \frac{1}{3}$
- 4) $RAPFD(\sigma_1, 4) = \frac{13}{32} < RAPFD(\sigma_2, 4) = \frac{1}{2}$
- 5) $RAPFD(\sigma_1, 5) = \frac{1}{2} < RAPFD(\sigma_2, 5) = \frac{3}{5}$

The overall results show us that: if testing resource constraint is greater than 3 (more than test cases could run), σ_2 detects faults more rapidly than σ_1 ; if testing resource constraint is less than 3 (less than 3 test cases could run), σ_1 and σ_2 have the same efficiency. And if testing resource constraint is 3 (just 3 test cases could run), though $RAPFD(\sigma_1, 3) = RAPFD(\sigma_2, 3)$, σ_2 is more efficient than σ_1 since the former detects more faults using first 3 test cases.

And for other two prioritized test suites $\sigma_3 : T_3 - T_2 - T_5$ and $\sigma_4 : T_3 - T_5$, their $RAPFD$ values for testing resource constraint $m = 1, 2, 3$ are:

- 1) $RAPFD(\sigma_3, 1) = RAPFD(\sigma_4, 1) = 0$
- 2) $RAPFD(\sigma_3, 2) = RAPFD(\sigma_4, 2) = \frac{3}{10}$
- 3) $RAPFD(\sigma_3, 3) = \frac{6}{15} < RAPFD(\sigma_4, 3) = \frac{8}{15}$

The overall results show us that: if testing resource constraint is greater than 2, σ_4 detects faults more rapidly than σ_3 ; if testing resource constraint is less than 2, σ_3 and σ_4 have the same efficiency. And if testing resource constraint is 2, though $RAPFD(\sigma_3, 2) = RAPFD(\sigma_4, 2)$, σ_4 is more efficient than σ_3 since the former detects more faults using first 2 test cases.

$RAPFD_C$, $RAPFD_W$, and $RAPFD_{CW}$ have the similar advantage, which is omitted here.

VII. CONCLUSION

We make a brief revisit of widely used existing $APFD$ series metrics, discuss the their physical explanations, and point out some limitations that may lead incorrect results especially in non-classic test case prioritization problems. To avoid limitations, a series of improved metrics are proposed in this paper. They could illustrate the process of faults detection in software testing more precisely and practically, and provide physical meaningful results to evaluate and compare the efficiency of prioritized test suites.

Besides the theoretical analysis and simple examples, more applications and case studies should be investigated in future works to examine the proposed metrics.

ACKNOWLEDGMENT

Supported by the National Natural Science Foundation of China (61300054), Natural Science Foundation of Jiangsu Province (BK20130879), Natural Science Foundation for Colleges & Universities of Jiangsu Province (13KJB520018).

REFERENCES

- [1] G. Rothermel, R. H. Untch, C. Y. Chu, M. J. Harrold. Prioritizing Test Cases for Regression Testing. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948.
- [2] X. Qu, M. B. Cohen, K. M. Wolf. Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization. In Proceedings of IEEE International Conference on Software Maintenance (ICSM2007): 255-264.
- [3] S. Elbaum, A. G. Malishevsky, G. Rothermel. Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization. In Proceedings of the International Conference on Software Engineering (ICSE2001): 329-338.
- [4] Dongjiang You, Zhenyu Chen, Baowen Xu, Bin Luo, Chen Zhang. An Empirical Study on the Effectiveness of Time-Aware Test Case Prioritization Techniques. In Proceedings of the 26th ACM Symposium on Applied Computing (SAC2011): 1451-1456.
- [5] R. C. Bryce, C. J. Colbourn. Prioritized Interaction Testing for Pairwise Coverage with Seeding and Constraints. Information and Software Technology, 2006, 48(10): 960-970.
- [6] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, R. S. Roos. Time-aware test suite prioritization. In Proceedings of International Symposium on Software Testing and Analysis (ISSTA2006), July 17-20, 2006 : 1-11.
- [7] S. WeiBleder. Towards Impact Analysis of Test Goal Prioritization on the Efficient Execution of Automatically Generated Test Suites Based on State Machines. In Proceedings of the 11th International Conference On Quality Software (QSIC2011), Madrid, Spain, July 13-14, 2011 :150-155.
- [8] M. J. Harrold, R. Gupta, and M. L. Soffa. A Methodology for Controlling the Size of A Test Suite. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270-285.

An Average Case Time Complexity Estimator for Black-box Functions

Duncan Yung, Bill Laboon, Shikuo Chang

Department of Computer Science, University of Pittsburgh

duncanyung@cs.pitt.edu, bill.laboon@pitt.edu, chang@cs.pitt.edu

Abstract—Average case time complexity is widely used to evaluate the efficiency of an algorithm [2]. Given a black-box function, if a tester wants to know the average case time complexity, he/she has to analyze the source code and make input assumption so as to know the average case of the function. Although that is feasible, it is time consuming and that makes the testing process no longer automatic. In this paper, we propose an approach for estimating the average case time complexity of a given black-box function without analyzing source codes. Experimental results show that our approach can accurately estimate the average case time complexity without reading the source code.

I. INTRODUCTION

The worst case time complexity is always used to evaluate the efficiency of an algorithm. The worst case time complexity of an algorithm is based on an extreme input which maximize the execution time of the algorithm. However, such extreme input may not always appear. Hence, average case time complexity can be a better representation of efficiency of an algorithm [2].

Given a black-box function, if a tester wants to know the average case time complexity, he/she has to analyze the source code and make input assumption so as to know the average case of the function. Although that is feasible, it is time consuming and that makes the testing process no longer automatic. Furthermore, manual source code analysis is not always reliable as it is not rare that human error occurs (That is one of the reason to have software testing.).

In this paper, we propose an approach for estimating the average case time complexity of a given black-box function without analyzing source codes. It is a useful tool for testers and programmers to analyze their source codes. The challenges of building such tools lie in two-fold:

- 1) Without reading the source code, we cannot get any hint from the implementation. The only easy thing that we can do is to measure execution time of the black-box function.
- 2) It is too time consuming to measure the execution time of all possible inputs so as to obtain the average execution time of the function and estimate the average case time complexity.

Related work study is in Section II. In Section III-A, we propose the baseline approach. Based on the baseline approach, we develop an advanced approach in Section III-C.

The accuracy and efficiency of different approaches are evaluated in Section IV.

II. RELATED WORK

The related work of time complexity analysis mainly fall into two areas-static time complexity analysis and measurement-based time complexity analysis. However, they all focus on worst-case time complexity. In this paper, we focus on average case time complexity.

A. Static Time Complexity Analysis

Static time complexity analysis approaches[9], [7] derive bounds for the execution time of a program without actually executing the program. Usually, the program is not treated as a black-box function and analysis of source codes is needed. In this paper, we propose a approach that can estimate the time complexity of black-box functions.

B. Measurement-based Time Complexity Analysis

Measurement-based approaches measures execution time of programs. As the number of possible inputs increase with the complexity of the program, exhaustive measurements becomes impossible. There exists solutions (e.g. [6] and [8]) that partition a program into parts for measuring execution time of worst case. These approaches try to bring the system into a worst-case state before taking measurements, e.g. by clearing the cache. However, this assumption may not hold for complex processor architectures that can exhibit timing anomalies. Kirner et al [8] proposed to generate inputs so that all paths of the program are taken. However, Kirner et al's approach is not a purely measurement-based time complexity analysis approach. Bernat et al [4], [3] proposed to determine the probability distribution of execution time. Their approach does not derive a bound for the execution time.

III. METHODOLOGY

In this paper, we propose to use execution time of the black-box function (Section III-A.1) and regression to estimate the average case time complexity (Section III-A.2) of a black-box function.

A. Baseline Approach

In this paper, we assume that the average case time complexity of a function is analyzed based on the uniform input assumption (Definition 1). That is the probability of the appearance of an input is always uniformly distributed over all possible inputs. For example, if the input is a size 3 integer array and only 1, 2, and 3 (domain size=3) are valid values for each element of the array, the probabilities that $\{1,1,1\}, \{1,1,2\}, \{1,1,3\}, \dots, \{3,3,3\}$ (there are 27 possible inputs) will be the input are the same and sum of the probabilities is 1.

Definition 1: [Uniform Input] Let x be an input. x satisfies: $\forall x \in \{0,1\}^*$, $prob(input = x) = \frac{1}{|\{0,1\}^*|}$, where $\{0,1\}^*$ is all possible bit patterns and $|\{0,1\}^*|$ is the size of the set $\{0,1\}^*$.

1) *Data Collection Phase:* Based on the uniform input assumption, we estimate the average execution time of an input size by measuring the average execution time of all possible inputs. Theoretically, we can generate all possible inputs and estimate the average execution time for $n=1, \dots, \infty$. By doing that, we can obtain data point of the average execution time of all input sizes.

2) *Prediction Phase:* Non-linear regression can be used to fit different curves to the data points. The best fit curve is the one with the least means square error. We adopt the best fit curve to be the average case time complexity of the function. For example, we use non-linear regression to fit the data points to functions $a + bn$, $a + b \log_2 n$, $a + bn \log_2 n$, $a + bn^2$, $a + bn^3$, and $a + bn^4$, where n is the input size, and a and b are coefficients. For each function, we can obtain the mean square error. The function with the least mean square error is chosen to be the average case time complexity estimation of the black-box function.

B. Bottleneck in Data Collection Phase

The number of all possible inputs increases exponentially with with input size. For example, the number of all possible inputs of a size 10 integer array is 10^{10} , given that the value of each element of the array can only be 1,2,...,10. Therefore, it is impossible to measure the execution time when input size is large. Although it is still possible to measure execution time of a function when input size is small (e.g. 5), the estimation of the average case time complexity is not accurate. (Figure 1, 2, and 3, Sol_3).

C. Sampling Approach with Majority Voting

In this paper, we propose to use uniform sampling of input to represent all possible inputs so as to improve efficiency of the model and majority voting technique to reduce variance of the estimation result.

1) *Efficient Sampling:* For each input size n , we draw samples uniformly from all possible inputs. The samples can be used as an estimation of all possible inputs as the probability of each input being drawn is the same.

2) *Majority Vote:* For each run, we run the model for k times, where k is a user-defined parameter. Then, we pick the majority prediction result as the result of that run. Suppose k is 10. There 6 times the prediction result is $a + bn \log_2 n$ and 4 times the prediction result is $a + bn^2$. Then, the prediction result of that run is $a + bn \log_2 n$.

In this setting, the majority voting is the same as bagging in machine learning field. It is well-known that bagging can successfully improve stability of models [5].

IV. EXPERIMENT

In this section, we evaluate the accuracy and efficiency of seven different approaches using 14 algorithms (black-box functions) which have integer array as inputs. We assume that the domain size of each integer is 1 to maximum integer in Java. Algorithm 1 to 10 are well-known algorithms. The definitions of algorithm 11 to 14 can be found in [1]. Experiments are implemented in Java and run in an Intel Core i5 2.5GHz laptop with 4G memory. All source codes can be found in https://github.com/duncanyung/cs1699Fall14_deliverable4.git. We compare accuracies and execution time of different approaches.

A. Comparison of Different Approaches

The settings of each approach are as below:

1) Sampling Approach with Majority Voting (Sol_1)

- input size $n=1,2,5,10,50,100$
- sample size for each $n=5000$ (sample size for algorithm 5 is 1000)
- majority voting of 50 predictions
- 50 runs

2) Sampling Approach **without** Majority Voting (Sol_2^a)

- input size $n=1,2,5,10,50,100$
- sample size for each **$n=5000$**
- 50 runs

3) Sampling Approach **without** Majority Voting (Sol_2^b)

- input size $n=1,2,5,10,50,100$
- sample size for each **$n=20000$**
- 50 runs

4) Sampling Approach **without** Majority Voting (Sol_2^c)

- input size $n=1,2,5,10,50,100$
- sample size for each **$n=80000$**

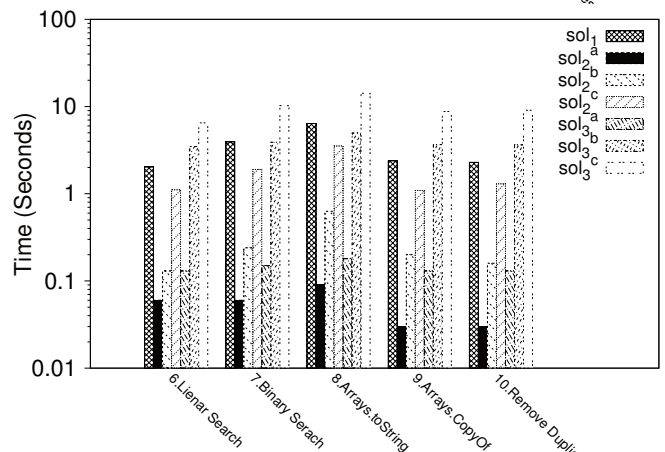
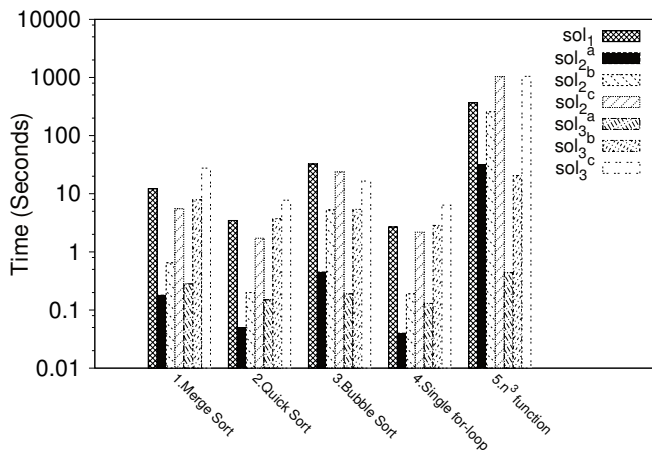
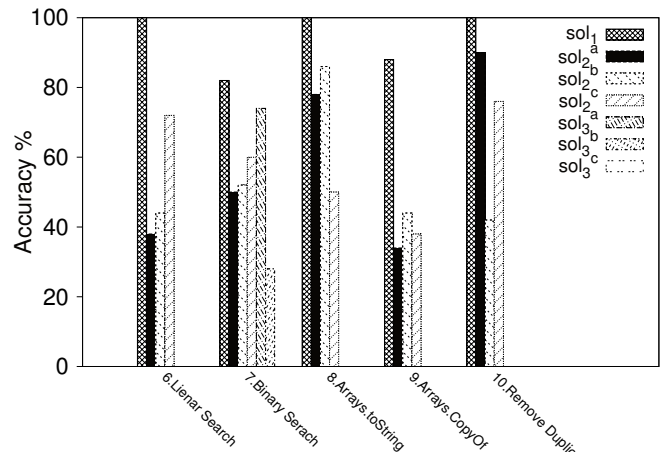
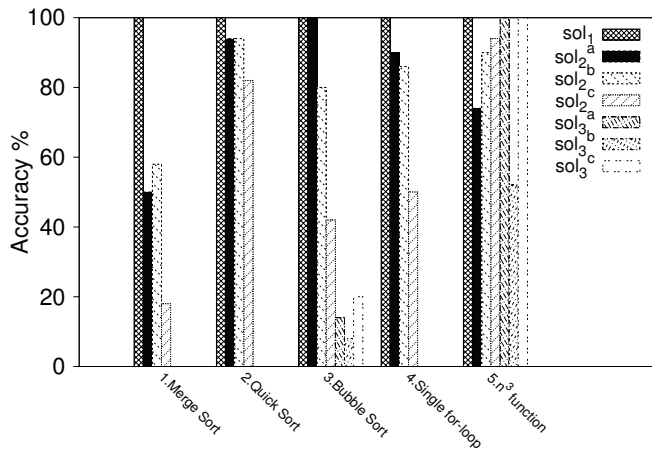


Fig. 1. Algorithm 1 - 5

Fig. 2. Algorithm 6 - 10

- 50 runs

5) Baseline Approach **with** Majority Voting (Sol_3^a)

- input size $n=1,2,3,4,5,6,7$
- majority voting of 50 predictions
- 50 runs

6) Baseline Approach **without** Majority Voting (Sol_3^b)

- input size $n=1,2,3,4,5,6,7,10$
- majority voting of 50 predictions
- 50 runs

7) Baseline Approach **with** Majority Voting (Sol_3^c)

- input size $n=1,2,3,4,5,6,7,10$
- majority voting of 50 predictions
- 50 runs

For each approach, the model tries to classify the average case time complexity of the black-box function as one of these time complexities- $O(n)$, $O(\log_2 n)$, $O(n \log_2 n)$, $O(n^2)$, $O(n^3)$, $O(n^4)$, and $O(n^5)$.

Figure 1, 2, and 3 shows the experiment results of all solutions for algorithm 1-5, 6-10, and 11-14 respectively.

The accuracy of Sol_1 is higher than Sol_2^a . Hence, we can see that the majority voting technique can actually improve the accuracy. The accuracy of Sol_3^a is the worst. Hence, we can see that using a small input size (from 1 to 7) cannot help to estimate the complexity. However, increasing input size will make the execution time increase exponentially which is not a feasible solution. Although the execution time of Sol_1 is the highest, the execution time of Sol_1^a is less than 200 seconds (except algorithm 5).

Sol_2^b and Sol_2^c tries to improve the accuracy by increasing the sample size for each input size to 20000 and 80000 respectively. In general, the accuracy of Sol_2^b is better than Sol_2^a , but with higher execution time. Although the accuracy of Sol_2^b is better than Sol_2^a , it is still worse than Sol_1 . Then, we further increase the sample size for each input size from 20000 to 80000. Hopefully, the accuracies would be improved. Unfortunately, the accuracy gets worse. Hence, we believe that using large sample size without majority vote cannot help improving accuracy.

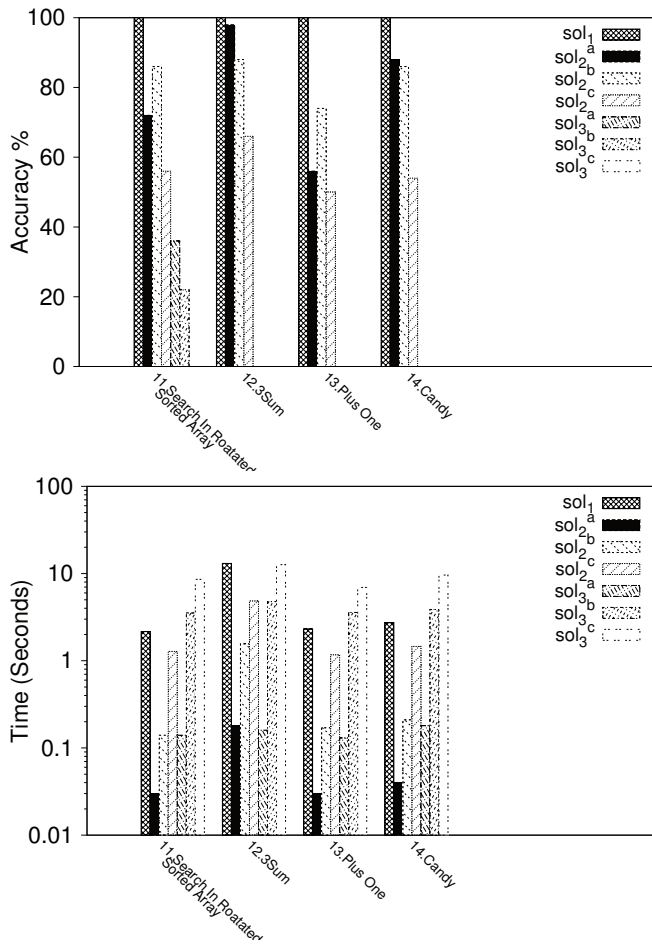


Fig. 3. Algorithm 11 - 14

Sol_3^b tries to improve the accuracy by changing the input size to $[1,2,3,4,5,6,7,10]$ while not using majority vote. Sol_3^c tries to improve the accuracy by changing the input size to $[1,2,3,4,5,6,7,10]$ while using majority vote. The accuracy of Sol_3^b and Sol_3^c are similar to Sol_3^a . However, there is a non-linear increase in average execution time. This shows that slightly increasing the input size (from $[1,2,3,4,5,6,7]$ to $[1,2,3,4,5,6,7,10]$) cannot help improving accuracy. However, largely increasing the input size (e.g. using $[1,2,5,10,50]$) would result in an unacceptably high execution time.

V. DISCUSSION

A. Execution Time Issue

The execution time of high complexity algorithms increase non-linearly (e.g. algorithm 5). Under such situation, the model may have to automatically reduce the input and sample size so as to have a quick response. Therefore, the model can estimate the execution time of the algorithm before deciding the input and sample size.

B. Input-Execution Time Relationship

In this paper, we assume that the execution time is related to the input size. However, that is not always true for an arbitrary black-box function. We need another approach for estimating the average case time complexity of those black-box functions. One of the direction to solve this issue is that the system can draw uniform input size sample instead of setting the input size to a specific sequence (e.g. $n=1,2,5,10,50,100$).

C. Input Type

In the experiment section, we assume that the input is an array of integer. However, the input of a black-box function can be any type. Therefore, generating uniform input for black-box functions with integer array as input is different from generating uniform input for black-box functions with other input types. Given a uniform input generator is the precondition of using our average case time complexity estimator.

VI. CONCLUSION

In this paper, we propose an approach to estimate the average case time complexity of a black-box function. We propose to use sampling to improve the efficiency and majority voting to improve the stability of the approach. Experimental result shows our proposed approach (Sol_1) can estimate the average case time complexity of different black-box functions accurately and efficiently.

REFERENCES

- [1] www.leetcode.com.
- [2] S. Ben-david, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44:193–219, 1997.
- [3] G. Bernat, A. Colin, and S. Petters. pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems. Technical report, 2003.
- [4] G. Bernat, A. Colin, and S. M. Petters. Wcet analysis of probabilistic hard real-time system. In *RTSS*, pages 279–288. IEEE Computer Society, 2002.
- [5] L. Breiman. Bagging Predictors. *Mach. Learn.*, 24(2):123–140, Aug. 1996.
- [6] J.-F. Deverge and I. Puaut. Safe measurement-based WCET estimation. In R. Wilhelm, editor, *5th International Workshop on Worst-Case Execution Time Analysis (WCET'05)*, volume 1 of *OpenAccess Series in Informatics (OASIS)*, Dagstuhl, Germany, 2007. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [7] S. Gulwani. Speed: Symbolic complexity bound analysis. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2009.
- [8] R. Kirner, P. Puschner, and I. Wenzel. Measurement-based worst-case execution time analysis using automatic test-data generation. In *IN PROC. IEEE WORKSHOP ON SOFTWARE TECH. FOR FUTURE EMBEDDED AND UBIQUITOUS SYSYS. (SEUS05)*, pages 7–10, 2004.
- [9] R. Wilhelm. Determining bounds on execution times.

Automatic Detection of Parameter Shielding for Test Case Generation*

Jingjian Lin^{1,2}, Jun Yan¹, and Jifeng Xuan^{3,4}

¹Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, China

²University of Chinese Academy of Sciences, China

³State Key Laboratory of Software Engineering, Wuhan University, China

⁴INRIA Lille - Nord Europe, France

Email: {linjingjian12,yanjun}@otcaix.iscas.ac.cn, jifeng.xuan@inria.fr

Abstract

Parameter shielding refers to the situation that one test parameter disables others in test execution. The quality of test case generation techniques is limited by the wide existence of parameter shielding. It is challenging to automatically find out conditions that cause the parameter shielding. This paper presents a novel approach for exploring the shielding conditions of test parameters. Our approach executes test inputs and collects runtime information of execution as features of test inputs. Then, a clustering algorithm is used to group test inputs with similar runtime information while a decision tree algorithm is built to extract the conditions in the groups. Finally, our approach identifies the shielding conditions based on the decision tree. Experiments on seven programs show that our approach can effectively detect the parameter shielding and the related conditions.

Keywords: black-box testing, parameter shielding, clustering, decision tree

1. Introduction

Software companies employ testing techniques as one indispensable step for quality assurance. A Software Under Test (SUT) has multiple input parameters and each parameter may lead to a large input space. In practice, it is expensive to verify the correctness of SUT using exhaustive testing [1], [2]. A variety of test case generation techniques have been proposed to reduce the scale of test cases, such as equivalence partitioning, boundary-value analysis, and category-partition methods [3].

*Corresponding author: Jifeng Xuan. This work is supported by National Natural Science Foundation of China (under grant No. 91118007) and INRIA Postdoctoral Research Fellowship.

In test case generation for a SUT with more than one parameter, one parameter may be shielded by others. *Parameter shielding* refers to the situation that one parameter disables other parameters in the SUT [4]. For example, if an application opens a modal child window, all operations to the parent window will be shielded; for many command-line tools in Linux system, the parameter `--help` shields all the other parameters. Parameter shielding results in the redundancy and low quality for test cases generation. For example, in combinatorial testing with Mixed Covering Array (MCA), parameter shielding will make test case generation fail in exposing potential errors, which should be detected if no parameter shielding exists [4]. Existing work by Chen et al. [4] focuses on how to generate combinatorial test cases under the scenario of shielding conditions. However, to the best of our knowledge, how to automatically detect the parameter shielding has been unexplored yet.

We propose an approach to automatic parameter shielding detection in this paper. This approach generates test inputs and extracts function call information via dynamic analysis tools. Cluster analysis groups inputs that show the similar function calls while a decision tree algorithm identifies constraints of parameters in clusters. By analyzing the result of the decision tree, we find out whether there exists parameter shielding and extract the condition that causes the parameter shielding. Experiments show that our approach can effectively detect the parameter shielding and the related conditions.

2. Background

2.1. Parameter shielding

Parameter shielding widely occurs in the case that several parameters control the same or relevant program logics [4]. However, in automatic testing, it is hard to be aware of

Table 1: 2-way test cases of the example
 Table 2: 2-way test cases under parameter shielding

test index	a	b	c	test index	a	b	c
t1	1	1	1	t1	1	1	1
t2	1	2	2	t2	1	2	2
t3	2	1	2	t3	2	1	#
t4	2	2	1	t4	2	2	#

parameter shielding before test case generation. This may fail to satisfy the requirements of a specific test case generation technique. Consider the following scenario of combinatorial testing, a SUT has three parameters a, b, and c with valid values of {1,2}. Table 1 shows the test cases generated by 2-way testing. The technique, *t*-way testing, aims to cover every possible combination value of no more than *t* parameters [4], [1].

Assume that parameter c is shielded by a: when a==2, c is disabled. Then test cases in Table 1 can be transformed into cases in Table 2 ('#' means the parameter is disabled). We find that test cases in Table 2 have not covered the following value pairs of b and c: <2, 1> and <1, 2>, which are originally covered in Table 1. In other words, when parameter a shields c, test cases cannot meet the requirements of 2-way testing. Therefore, it is important to find out the potential parameter shielding before test case generation.

2.2. Data mining

Data mining, aims to extract implicit, previously unknown, and potentially useful information from databases [5]. It is actually the process of finding the hidden data pattern of the databases.

In this paper, we leverage clustering and classification techniques to analyze runtime logs. The goal of clustering is to discover similarities and differences among data patterns in order to derive useful conclusions about similar clusters [6]. According to a specific similarity measure, a data set is divided into clusters; such division ensures that data inside one cluster have a higher similarity than those in different clusters [7].

The goal of classification is to predict categorical labels, such as “safe” or “risky” for the loan application data, “yes” or “no” for marketing data [7]. Decision trees are a typical family of classifiers on a target class in the form of a tree structure. One main advantage of decision trees is to produce a set of rules, which represents the branches and nodes of the tree; such rules can be easily interpreted into condition combinations, comparing with other basic classification techniques [8].

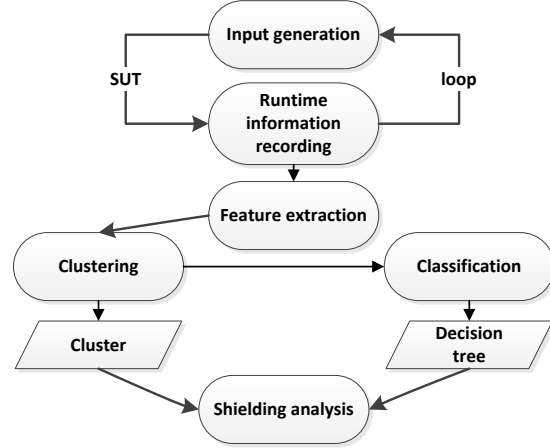


Figure 1: Process of parameter shielding detection

3. Approach

We propose an approach to automatically detecting parameter shielding. Figure 1 shows the process of our approach, which consists of six steps.

3.1 Input generation

In general, parameter shielding occurs when a parameter value dominates a module or several parameters are sensitive to the parameter orders. In our work, to detect parameter shielding, we extract parameters in the same module and generate test inputs for further test execution.

First, we generate test inputs, i.e., input vectors of actual values for parameters, for every single parameter. We can generate all values in the input space for parameters with small input space. For parameters with large input space, we choose parameter values that generate different runtime behaviors, such that these values can be divided into different clusters by clustering analysis. Values can be selected by equivalence partitioning, boundary-value analysis etc. Based on these selected values, we calculate all combination and permutations of parameters. So the number of generated test inputs is $n! \times \prod_{i=1}^n v_i$, where n denotes the number of parameters and v_i denotes the number of values for the i th parameter.

3.2 Runtime recording and feature extraction

We execute all generated inputs and extract their runtime information. Using dynamic analysis tools such as Valgrind [9],¹ we are able to collect running information of SUTs, including the orders of function calls and the number of function calls. Then, function call information is stored in logs for feature extraction.

¹<http://valgrind.org/>

We describe the two kinds of features in our experiments as follows. The *order of function calls* refers to the index of a function such as `bar` in the call chain of another function such as `foo` during the test execution. For example, if `foo` calls 10 functions in one execution and `bar` is the second function called by `foo`, then the order of `bar` called by `foo` is 2. Similarly, the number of function calls refers to the count of a function such as `bar` called by `foo` during execution. For example, if `foo` calls `bar` nine times, then number of function calls of `bar` called by `foo` is 9.

```

1 def gen_func_call_pair(logs):
2     call_dict = dict()
3     for func in calling_functions:
4         for subfunc in functions_called_by_func:
5             if not call_dict[func]:
6                 call_dict[func] = set()
7                 call_dict[func].add(subfunc)
8     return call_dict
9 def gen_numeric(call_dict, log):
10    features = []
11    for func in call_dict:
12        for subfunc in call_dict[func]:
13            if subfunc is called by func in log:
14                feature func:subfunc = its calling order
15            else:
16                feature func:subfunc = -1
17            features.append(func:subfunc)
18    return features
19 def gen_featMat(logs):
20    call_dict = gen_func_call_pair(logs)
21    for log in logs:
22        features = gen_numeric(call_dict, log)
23    print features to file

```

Listing 1: Feature extraction of the orders of function calls

The order of function calls can be converted to numeric features by the python-like pseudo code in Listing 1. Function `gen_func_call_pair` scans logs that are record by dynamic analysis tools and finds out the collection of functions which called by the same function. Then we can obtain all function call pairs. For example, `funcA:funcB` means the order of `funcB` in the collection of functions which are called by `funcA`. Function `gen_numeric` is used for generating the number value of specific features. It scans logs that record calling information of a specific input and calculates the feature values. Note that some call pairs only occurs in some specific inputs. Thus, we assign a specific value such as -1 to values of call pairs which are not occurred. Function `gen_featMat` calls `gen_func_call_pair` and `gen_numeric` to generate all values of features and write them to file. The feature extraction for the number of function calls is similar to the feature extraction for the order of function calls.

3.3. Clustering and decision tree algorithms

Based on the extracted features, we apply clustering algorithms based on numeric distances to detect similar test inputs [7]. Clustering, such as k-means and EM algorithms [6], is used in our approach for grouping inputs which

generate similar program behaviors together in a cluster. In our work, the k-means algorithm is used for clustering in the implementation. Since we select values that conduct different program behaviors, the inputs will be grouped into different clusters. Recall the example in Table 1, assume that all parameter values lead to different program behaviors. Then case $\langle 2, 1, 1 \rangle$ and $\langle 2, 1, 2 \rangle$ will be grouped into different cluster. If `c` is disabled when `a==2`, then case $\langle 2, 1, 1 \rangle$ and $\langle 2, 1, 2 \rangle$ will be grouped in the same cluster, called `cluster1`. This cluster (`cluster1`) is determined by value of `a` and `b` only, meanwhile other clusters are determined by values of all parameters. We can find out the shielding condition by analyzing the clustering result.

To further “understand” clusters, we employ a decision tree algorithm to find out the conditions for parameter shielding. Given clusters as well as their test inputs, we treat a cluster, which a test input belongs to, as the label of the test input. Then we have a data set of labeled test inputs. Based on this data set, we train a decision tree model to build the relationship between features and their labels (clusters). Decision trees are a typical kind of classification algorithms, for example, ID3, C4.5, and CART [10]. C4.5 is used in our implementation.

3.4. Shielding analysis

When a parameter is shielded by others, the parameter value will not affect the program behavior. In other words, when a parameter is disabled, runtime behaviors of the SUT are only determined by other parameters.

We can find the shielded parameter from decision tree by the following steps: first, in a decision tree, we merge paths (conditions) that come from the same cluster; second, we find out parameters that do not exist in the conditions of clusters. These parameters are the shielded parameters while the identified conditions could be the shielding conditions.

4. Experiments

Experiments are conducted on programs of different types and scales. Table 3 lists seven programs in our experiments. The experiments are implemented with open-source data mining platforms, Weka.²

4.1 Program with enumeration inputs

Taking program `ln` as an example, three parameters `-s`, `-P`, and `-L` are considered in the experiment. Parameter `-s` is used for making symbolic links instead of hard links; parameter `-P` is used for changing hard links directly into symbolic links; and parameter `-L` is for changing hard links into symbolic link references.

²<http://www.cs.waikato.ac.nz/ml/weka/>

Table 3: Seven SUTs in experiments

	Name	#Parameter	#Feature	#Input	Kloc	Description
1	ls	3	366	12	5.0	print a list of the current directory
2	cp	3	318	4	1.2	copy files or directories
3	ln	3	56	12	0.6	establish link to a file or directory
4	head	2	82	800	1.0	display the first few lines of a file
5	tail	2	108	800	2.3	display the last few lines of a file
6	bzip2	2	182	4	7.3	a compression program
7	ffmpeg	2	6358	200	892.5	solution to audio and video processing

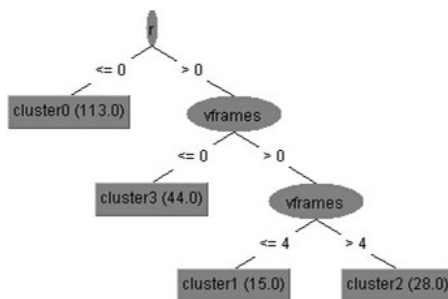
Table 4 shows the classification result of `ln`. We can find out that test inputs of `-L` and `-PL` belong to the same cluster (`cluster1`), while inputs of `-P` and `-LP` belongs to another cluster (`cluster2`). We can infer that when `-P` is in front of `-L`, program behaviors are almost the same as input `-L`. Thus, we conclude that `-P` is shielded if `-P` is listed in front of `-L`. Similarly, `-L` is shielded when `-L` is in front of `-P`. Meanwhile, all inputs that contain the parameter `-s` belong to the same cluster (`cluster0`), so we can conclude that `-P` and `-L` are shielded when `-s` exists in the inputs. We confirm the above conclusions by manually verification.

4.2 Program with integer inputs

We discuss shielding of integer inputs in this subsection. Figure 2 shows the decision tree of parameter `-vframes` (`<number>`) and `-r` (`<fps>`) of `ffmpeg`. Parameter `-r` set the number of video frames to output and parameter `-vframes` set frame rate of the input video. Since the path of `cluster0` is not divided by `vframes`, we can conclude that when `r ≤ 0`, `vframes` is shielded. By manually executing the program, we confirm that the conclusion is correct.

Table 4: Classification result of `ln` with parameters `-L`, `-P`, and `-s`

cluster0	cluster1	cluster2
-s,-Ls,-sL,-Ps,-sP	-L	-P
-LPs,-LsP,-PLs	-PL	-LP
-PsL,-sPL,-sLP		

Figure 2: Classification result of `ffmpeg`Table 5: Classification results of `cp`, `head`, and `bzip2`

Name	Parameter	Cluster	Condition
ls	-g,-A	cluster0 : -g, -gA, -Ag cluster1 : -A	-g shields -A
cp	-s,-L	cluster0 : -s, -sL, -Ls cluster1 : -L	-s shields -L
head/tail	-n(lines) -c(bytes)	cluster0 : c ≤ 0 cluster1 : 0 < c ≤ 20 cluster2 : c > 20	-c shields -n
	-c (bytes) -n(lines)	cluster0 : n ≤ 0 cluster1 : 0 < n ≤ 9 cluster2 : n > 9	-n shields -c
bzip2	-t,-d	cluster0 : -t, -dt cluster1 : -d, -td	later parameter shields the former

4.3 Results for other programs

Table 5 shows other results of SUTs in Table 3. For all these programs, we find that the shielding conditions are correctly detected.

5. Conclusion

This paper proposes a novel approach to automatic detection of parameter shielding for test case generation. Test parameters shielded by others are found by clustering the runtime information of the SUT. Experiments show that our approach can effectively detect the parameter shielding for various types of parameters. Meanwhile, shielding conditions between parameters can also be detected in our approach. Our shielding detection approach can be used for enhancing the quality of test cases and for reducing the testing cost.

References

- [1] J. Zhang, Z. Zhang, and F. Ma, *Automatic Generation of Combinatorial Test Data*. Springer, 2014.
- [2] J. Xuan and M. Monperrus, "Test case purification for improving fault localization," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 52–63, ACM, 2014.
- [3] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 11, 2011.
- [4] B. Chen, J. Yan, and J. Zhang, "Combinatorial testing with shielding parameters," in *Software Engineering Conference (APSEC)*, pp. 280–289, 2010.
- [5] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, p. 37, 1996.
- [6] T. J. Oyana, "A new-fangled fes-k-means clustering algorithm for disease discovery and visual analytics," *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2010, no. 1, p. 746021, 2010.
- [7] M. Kantardzic, *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons, 2011.
- [8] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender systems handbook*, vol. 1. Springer, 2011.
- [9] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in *ACM Sigplan Notices*, vol. 42, pp. 89–100, ACM, 2007.
- [10] S. Ruggieri, "Efficient c4. 5 [classification algorithm]," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 14, no. 2, pp. 438–444, 2002.

PIPE+Verifier - A Tool for Analyzing High Level Petri Nets

Su Liu and Xudong He

School of Computing and Information Sciences

Florida International University

Miami, Florida 33199, USA

{sliu002, hex}@cis.fiu.edu

Abstract—High level Petri nets (HLPNs) have been widely used to model complex systems; however, their high expressive power costs their analyzability. Model checking techniques have been exploited in analyzing high level Petri nets, but have limited success due to either undecidability problem or state explosion problem. Bounded model checking (BMC) is a promising analysis method that explores state space within a predefined bound. BMC sacrifices the completeness of traditional model checking but becomes more practical and often effective to analyze large models. In our prior work, we have developed a method based on BMC and a supporting tool PIPE+Verifier to analyze high level Petri nets using a state of the art satisfiability modulo theories (SMT) solver Z3 as the backend engine. Our experiment results have been very encouraging. In this paper, we present the design, implementation, and use of PIPE+Verifier, as well as show additional improvements to make PIPE+Verifier more efficient.

Keywords- Petri Net, Model Checking, Bounded Model Checking.

I. INTRODUCTION

High level Petri nets (HLPNs) [2] have been widely used to model the data, functionality, structure, and dynamic behaviors of complex systems. However the powerful expressiveness of HLPNs costs their analyzability. Simulations are the primarily analysis technique for HLPNs. Many HLPNs modeling tools such as CPN tools [10], [1], ALPiNA [9] and PIPE+ [12] support the simulation of different forms of HLPNs. While simulation is practical and cost effective, it cannot assure a safety property to be satisfied in all possible executions. Exhaustive analysis methods such as model checking [11] search all possible execution paths of a model but suffer from the state explosion problem, and are often limited to finite state systems. Since HLPNs can be used to model complex systems, where the state space can be not only huge but also infinite.

Bounded model checking (BMC) with satisfiability solving [6], [3] was proposed as an alternative approach to address the state explosion problem, which is particularly suited to analyze safety properties. BMC tries to find a counterexample violating a safety property by exploring only a finite state space defined by all execution paths up to a pre-defined bound k . A counterexample is found if the negated safety property is held in a reachable state; otherwise, the safety property holds

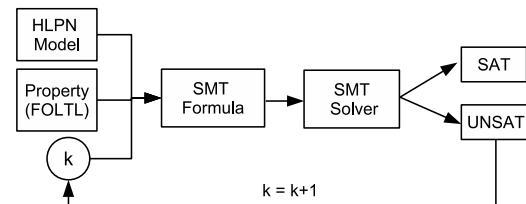


Figure 1: An Overview of PIPE+Verifier's Workflow

up to k . k can be iteratively increased to an acceptable value proportional to the size of a model's state space. Since the true upper bound cannot be determined in general, BMC is not a complete analysis method, yet is practical and effective in many real-world applications.

In SAT-based BMC [5], a model is converted into a propositional formula whose satisfiability is determined by a SAT solver. In recent years, satisfiability modulo theories (SMT) solvers [7] have made great progresses to efficiently check the satisfiability of a subset of first-order logic formulas with a variety of underlying theories including linear arithmetic, difference arithmetic, arrays and so on. These theories are rich enough to represent the data and algebraic expressions in most HLPN models.

In [13], we have developed a method based on BMC and a supporting tool PIPE+Verifier to analyze high level Petri nets using a state of the art satisfiability modulo theories (SMT) solver Z3 [8] as the backend engine. We have applied PIPE+Verifier to a variety of models from the existing literature and obtained very encouraging experimental results. An overview of PIPE+Verifier's workflow is shown in Figure 1.

PIPE+Verifier has the following features:

- being compatible with HLPN models and first-order linear time logic (FOLTL) representing properties built by modeling tool PIPE+ [12];
- encoding HLPN models and safety properties in FOLTL into an SMT formula;
- exporting the SMT formula into a file in C language (written in Z3's C API) recognizable by Z3;
- invoking Z3 to check the satisfiability of the SMT formula and returning an analysis report with the checking result, counterexample, consumed time and memory to check; and
- allowing incremental checking by increasing the k upper

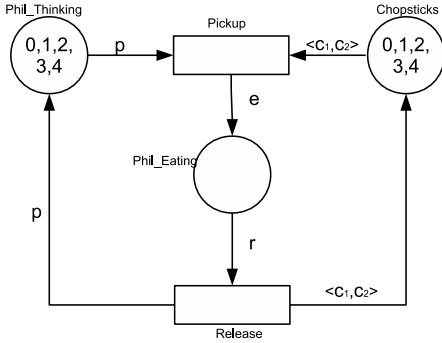


Figure 2: Five Dining Philosopher Problem in HLPN

bound value.

In this paper, we present the design, implementation, and use of PIPE+Verifier, as well as show additional improvements to make PIPE+Verifier more efficient with some experimental results.

II. BACKGROUND

A. High Level Petri Nets

A HLPN [2] has a net structure consisting of a finite set of places (drawn as circles), a finite set of transitions (drawn as bars), and a finite set of directed arcs between places and transitions (drawn as arrows); and a net inscription supporting the definitions of place types, place markings, arc annotations, and transition conditions. A place type can be a power set to capture a set of tokens. All the tokens in a power set are of the same type built from primitive data types including integer type and string type. A place marking is a collection of tokens (data items) associated with the place. Arc annotations are inscribed with expressions that may comprise constants, variables, and function images. Transition conditions are logic expressions.

Figure 2 illustrates a dining philosopher problem modeled in a HLPN. The net consists of three places $P_{Phil_Thinking}$, $P_{Chopsticks}$, P_{Phil_Eating} and two transitions T_{Pickup} and $T_{Release}$. All the places' token type is $\langle int \rangle$. $P_{Phil_Thinking}$ and $P_{Chopsticks}$ both have five tokens initially $\{\langle 0 \rangle, \langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle 4 \rangle\}$. T_{Pickup} 's transition condition is $p = c_1 \wedge (p+1)\%5 = c_2 \wedge e = p$. $T_{Release}$'s transition condition is $p = r \wedge c_1 = r \wedge c_2 = (r+1)\%5$.

B. Bounded Model Checking

Different from traditional model checking, BMC is incomplete and only performs an exhaustive search up to an upper bound. In many real world applications, a property can be effectively checked by examining only the limited prefixes of all executions, thus BMC becomes a practical and useful analysis technique, which partially alleviates the state explosion problem. Given a finite transition system M , a linear time temporal logic (LTL) formula f , and an integer k ; BMC tries to determine whether there exists a computation path in M of length k or less (denoted as M_k) that satisfies f .

In BMC, a logic formula ϕ_k is constructed from a given M_k , including the initial state I and unrolled transition relation T

and some properties f . Since transition T in ϕ_k is unrolled k times, the length of ϕ_k is dependent on k . The logic formula ϕ_k is represented in equation 1:

$$\phi_k \doteq I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^k \neg f(s_i) \quad (1)$$

where $I(s_0)$ is the characteristic function of the initial state, $T(s_i, s_{i+1})$ is the characteristic function of the transition relation, and $f(s_i)$ represents the property formula f associated with unrolled state s_i ($0 \leq i \leq k$). Currently our tool supports the analysis of safety properties, thus f represents some safety property. If ϕ_k is satisfiable, there is a firing sequence or a state transition path from the initial state $I(s_0)$ to a state s_i that satisfies the negation of f_i , thus violates f ; otherwise, property f holds in all execution sequences up to k transition steps in M .

Satisfiability modulo theories (SMT) [7] solvers are efficient modern theorem provers that support a combination of underlying theories such as bit-vectors, rational and integer linear arithmetic, arrays, and uninterpreted functions. SMT solvers are the extensions of satisfiability (SAT) solvers and directly applicable to the decision problems expressed in first order logic formulas with respect to the multiple background theories. For example, an SMT solver can decide whether a formula in the theory of linear arithmetic is satisfiable:

$$(x + y \leq 0) \wedge (\neg b \vee a \wedge (y = 0)) \wedge (x \leq 0)$$

where x, y are integer variables and a, b are Boolean variables. If the formula is satisfiable, the SMT solver returns a variable assignment satisfying the formula.

Both SAT solvers and SMT solvers have been successfully used in BMC. Z3 [8], developed in Microsoft Research Institution, is an efficient and widely used SMT solver that supports many background theories, such as rational and integer arithmetic, bit-vectors, array theory, and set theory. Z3 ranks highly in annual SMT competitions [4]. Therefore, Z3 is chosen as PIPE+Verifier's backend engine.

III. TRANSLATING HLPN MODELS TO SMT FORMULAS

A. General Translation Rules HLPNs to a SMT Formulas

In BMC [6], a model and a property are encoded into a formula ϕ_k , which is solved by a SAT or SMT solver. Encoding a HLPN model and a property formula into ϕ_k involves the following steps.

1) *Representing HLPN Markings as Symbolic States:* In a HLPN model, a marking M_i is defined by a distribution of tokens in all places. Thus we need to define a symbolic state in SMT covering all places and their types in the HLPN. A mapping from HLPN model's elements to SMT sorts is shown in Table I.

A marking is defined by a SMT tuple with each tuple element denoting a place in the HLPN. Since a place can contain multiple tokens, it is defined as a set in SMT. Structured token types defined in the HLPN are mapped to tuples in SMT, and primitive token types such as integer and strings are encoded as Integer in SMT.

Table I: HLPN Elements to SMT Sorts

HLPN Elements	SMT Sorts
Marking	Tuple
Places	Set
Structured Token Type	Tuple
Primitive Token Type	Integer

Since an execution sequence having k transition firing steps $M_0 \rightarrow M_1 \rightarrow \dots \rightarrow M_k$ contains $k + 1$ markings, which correspond to $k + 1$ symbolic states in ϕ_k , $k + 1$ sets of unique variables $\{V_0, V_1, \dots, V_k\}$ in SMT are needed.

2) *Encoding Initial State* : In a HLPN model, the initial state is defined by the initial marking. In ϕ_k , an initial symbolic state is defined by assigning values to the first symbolic state through clauses. The clauses are mainly expressed in equations. Thus a formula representing the initial marking in the HLPN is first constructed and then added as a conjunct to ϕ_k .

3) *Formulating Transitions*: In a HLPN model, each transition t_j captures a local state change and its firing subtracts tokens from t_j 's input places and adds tokens into t_j 's output places, which contributes to the overall marking change $M_i \rightarrow M_{i+1}$. The effect of firing each transition is encoded as a logic formula $t_j(S_i, S_{i+1})$. More specifically, let P be the set of places in HLPN model and p_{t_j} be the set of places connected to t_j , the relation is encoded as Equation 2.

$$t_j(S_i, S_{i+1}) = \begin{cases} S_{i+1}(p_{t_j}) = t_i(S_i(p_{t_j})) \\ S_{i+1}(P \setminus p_{t_j}) = S_i(P \setminus p_{t_j}) \end{cases} \quad (2)$$

Concurrent transition firings in the HLPN model need to be linearized, which does not affect the safety properties to be analyzed. Thus only interleaved executions are considered. Due to the non-determinism of transition firings, each firing (transition) step is encoded as a formula $T_i(S_i, S_{i+1}) = \bigvee_{j=0}^n t_j(S_i, S_{i+1})$ representing the disjunction of the formulas capturing the effects of firing individual transitions t_j ($0 \leq j \leq n$). An execution consisting of k successive state transition formulas as follows, which is added as a conjunct to ϕ_k :

$$\bigwedge_{i=0}^{k-1} \left(\bigvee_{j=0}^n t_j(S_i, S_{i+1}) \right) \quad (3)$$

4) *Defining Property* : In a HLPN model, properties are defined in FOLTL formula f . Since BMC is most effective in checking the violation of safety properties, a formula $f(S_i)$ representing the safety property formula f without temporal operators in state S_i needs to be checked. Formula $\bigvee_{i=0}^k \neg f(S_i)$ expressing the violation of the safety property in the first k transition step in an execution sequence is added as a conjunct to ϕ_k .

B. Specific Translation Code from HLPNs to Z3

PIPE+Verifier processes an HLPN model and translates it into C API code provided by Z3 solver so that Z3 solver can compile and execute the code.

The generated C API code contains five parts:

Table II: SMT Declaration Z3 Code

SMT Sort	Code Example	Relation to HLPN
StateTUPLE	Z3_mk_tuple_sort()	Marking
PlaceSetSORT	Z3_mk_set_sort()	Place
TokenSORT	Z3_mk_tuple_sort()	Structured token type
IntegerSORT	Z3_mk_int_sort()	Primitive token type

- 1) **Declaration**: declares a list of required types (called SORT in SMT), shown in Table II.
- 2) **Defining symbolic states**: the state builder defines $k + 1$ states in C code, each state has a type STATETUPLE. The C code uses $Z3_ast\ S_i = Z3_mk_const(STATE_TUPLE)$, where S_i is the identifier of state i .
- 3) **Building initial state**: since symbolic states are defined, a formula capturing the initial marking asserts that an empty place set equals to S_0 . A code snippet is shown in Code 1:

Code 1: Initial State in Z3

```

1 Z3_ast ini_token_clauses[m];
2   Z3_ast ini_place = Z3_mk_empty_set(TokenSORT);
3   Z3_ast ini_token = Z3_mk_const(TokenSORT);
4   Z3_mk_set_add(ini_place, ini_token);
5   ...
6   Z3_ast ini_token_clauses[0] =
7     Z3_mk_eq(mk_unary_app(proj_decls[0], S0), ini_place);
8   ...
9   Z3_assert_cnstr(ctx, Z3_mk_and(m, ini_token_clauses[0]));

```

- 4) **Formulating transitions**: the formula is defined in terms of a conjunction of k successive state transitions $T(S_i, S_{i+1})$, where each state transition is defined by a disjunction of local transitions $t(S_i, S_{i+1})$. $t(S_i, S_{i+1})$ is defined by an *if-then-else* structure *if c_0 then c_1 else c_2* , which is a concise representation of $(c_0 \implies c_t) \wedge (\neg c_0 \implies c_f)$. A code snippet is shown in Code 2:

Code 2: Transition Formulation in Z3

```

1 Z3_ast transitions_state[k];
2   Z3_ast transitions_local_or[1];
3   Z3_ast var_in = Z3_mk_const(Z3_mk_set_sort(TokenSORT)
4     );
5   ...
6   Z3_ast cond_in = Z3_mk_set_member([input arc variable
7     ], [input place]);
8   Z3_ast cond_trans = [transition formula];
9   Z3_ast cond_out = Z3_mk_set_member([output arc
10    variable], [output place]);
11  Z3_ast cond = Z3_mk_and(o, cond_and);
12  Z3_ast trans_true_and[m];
13  ...
14  Z3_ast trans_true = Z3_mk_and(m, trans_true_and);
15  Z3_ast trans_false_and[n];
16  ...
17  Z3_ast trans_false = Z3_mk_and(n, trans_false_and);
18  Z3_ast transitions_local[0] = Z3_mk_ite(cond,
19    trans_true, trans_false);
20  ...
21  Z3_ast trans_dump = Z3_mk_eq(S0, S1);
22  Z3_ast transitions_local_dump = Z3_mk_implies(
23    Z3_mk_true(), trans_dump);
24  ...
25  Z3_ast transitions_state[0] = Z3_mk_or(1,
26    transitions_local_or);
27  ...
28  Z3_assert_cnstr(ctx, Z3_mk_and(k, transitions_state));

```

- 5) **Defining properties**: each safety property f is defined by a disjunction of negated formulas in successive

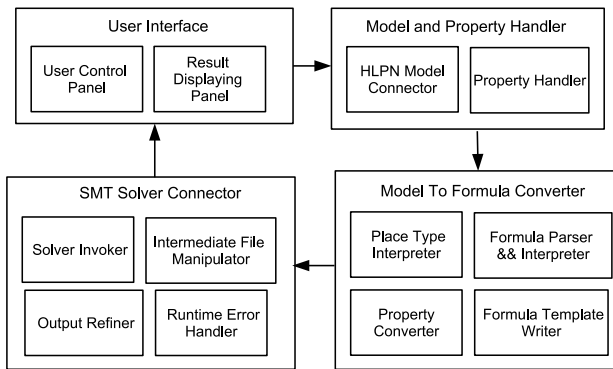


Figure 3: The Design View of PIPE+Verifier

states $\neg f(s_i)$. Thus a bad state indicated by a specific token reaching a particular place is checked using `Z3_mk_set_member()`. A code snippet is shown in Code 3:

Code 3: Property Definition in Z3

```

1  Z3_ast properties[k];
2  Z3_ast token = Z3_mk_const(TokenSORT);
3  ...
4  property[0] = Z3_mk_set_member(token, mk_unary_app(place,
5  S0));
6  Z3_assert_cnstr(Z3_mk_or(k, properties));

```

IV. PIPE+VERIFIER

PIPE+Verifier is developed as an additional analysis component of PIPE+ [12]. PIPE+ is a graphical HLPN editor and simulator. A user builds a HLPN model in PIPE+ by dragging and dropping graphical elements as well as editing specifications inside the graphical elements. PIPE+Verifier leverages this editor as an input source of HLPN models and launches PIPE+Verifier to conduct BMC on the models. Similar to PIPE+, PIPE+Verifier is implemented in Java, thus is able to run on any platform that can run Java Virtual Machine and Z3 solver. A detailed design view of PIPE+Verifier is shown in Figure 3:

1) *User Interface*: User interface in PIPE+Verifier is built as a dashboard (in Java Swing) that can take in user’s input for properties in FOLTL and command to start the checking process. Furthermore, it displays the analysis results as well as error messages that may encounter during the checking process.

Properties are built in a standard format in order to conform to the safety property that BMC can check effectively. The format eases the transformation of FOLTL formula into SMT formula. In order to restrict the user’s input format and specify the targeted bad state, a user is provided with a list of places fetched from the HLPN model and tokens need to be constructed according to the selected place types. A user needs to provide an upper bound k value required by BMC.

If a safety property holds for all states searched from the initial state to a depth up to predefined k , the displayer will show a message “SAT” and a resources consumed summary including time and memory usage; otherwise, a message

“UNSAT” is shown and a counterexample leading to the bad state is printed.

2) *Model and Property Handler*: Model and property handlers are used to prepare for the next model to formula converting process. Because PIPE+Verifier uses HLPN model built from PIPE+ model editor, the model connector is built to refine the HLPN model from PIPE+ and check the consistency of the model to avoid conversion error. The property handler, on the other hand, takes the user input property from the interface and prepares its conversion to an SMT formula.

3) *Model To Formula Converter*: Model To Formula Converter is the component to conduct this conversion process. The converter consists of several components including State Builder, Place Type Recognizer, Transition Parser and Interpreter, Property Converter and Formula Template Writer.

- *State builder*: The state builder defines a STATETUPLE (shown in the SMT context above) according to the structure of the HLPN model. The tuple is a structure that consists of a list of place sorts, each place sort is also a tuple. The complete state list for the SMT formula in Equation 1 contains $k + 1$ states.
- *Place type recognizer*: the recognizer traverses all the places in the HLPN model and stores all the distinct place types in order to construct distinct sorts in SMT context;
- *Transition Parser and Interpreter*: in the HLPN model, transition formula is a first-order logic formula that guards the token flow in the model. The formula needs to be parsed and interpreted into an abstract syntax tree in order to allow this tool to understand the first-order logic formula and build a corresponding SMT formula;
- *Property Converter*: in BMC, SMT solver’s responsibility is to search for bad state, which is satisfiable solution to the negated properties. The safety properties prepared by property handler is converted into a negated SMT formula by Property Converter. The converting process is straightforward.
- *Formula Template Writer*: The template writer leverages a predefined template file in C language that contains necessary utility functions for checking the SMT formula in Z3 solver, and fills in model and property information in order to build the input file that conforms to Z3 solver’s input format. Formula Template Writer writes the converted SMT formula’s declarations, states, transitions, and properties into the file that can be compiled and checked by Z3 solver.

4) *SMT Solver Connector*: The SMT Solver Connector handles the process of delivering the SMT formula to Z3 solver as well as receiving the checking result from Z3 solver. The connector consists of four components:

- *Solver Invoker*: the invoker links to the tool to the backend engine Z3, which contains some scripts to automatically launch Z3 with proper parameters. The scripts are shell scripts for Windows and Unix in order to allow the tool to run on different platforms.
- *Intermediate file manipulator*: As the analysis process includes conversions and checkings, intermediate temporary files are created such as a file to represent formula,

a compiled Z3 checker, a file to record Z3's checking result, and a file to store final result. Intermediate file manipulator is in charge of the creating and deleting temporary files.

- Runtime error handler: Since PIPE+Verifier involves an external tool and file system interactions, unexpected errors can happen, an error handler can prevent the tool get into failure and can better managing errors.
- Output refiner: the raw result generated by Z3 solver is not readable as it only checks the satisfiability of the intermediate SMT formula instead of the original model and property. Thus, it is necessary to rebuild the model and property's checking result based on Z3 generated result. Output refiner can process the Z3 result file by removing redundant information and reorganize structure, and present readable results to the user.

V. AN IMPROVED TRANSITION FORMULATION

The naive transition formulation given in the previous sections results in a ϕ_k capturing all the possible interleavings of transition firings in the given HLPN within depth k without considering the dependencies among them. The computation complexity of the naive method is thus exponential and is not computable with a large k value.

The firing of a transition depends on the existence of tokens from its input places P , thus depends on the other transitions producing tokens for P . If in a state s , a transition t 's input places are empty or do not have enough tokens to enable t , t cannot fire at state s . If in a state s , a transition t 's output places are not relevant to a given property, a transition firing sequence σ_k has a t as the last transition has no impact on the satisfiability of the property. With these observations and analysis, it is possible to build a more concise formula to avoid redundant checking by an SMT solver and thus improve the efficiency of BMC.

For example, in Figure 4, the initial marking is $P_0 \{tok_0\}$, $P_1 \{\}$, $P_2 \{\}$, if we want to check whether it can reach a marking where $P_2 \{tok_0\}$. The model formula produced by equation 1 with $k = 2$ is:

$$\phi_k = I(s_0) \wedge (t_i(s_0, s_1) \vee t_o(s_0, s_1)) \wedge (t_i(s_1, s_2) \vee t_o(s_1, s_2)) \wedge (\neg f(s_0) \vee \neg f(s_1) \vee \neg f(s_2)) \quad (4)$$

This formula ϕ_k covers all possible transition firing orders including $t_i \rightarrow t_i$, $t_i \rightarrow t_o$, $t_o \rightarrow t_i$ and $t_o \rightarrow t_o$. There are infeasible firing sequences in this net model. Firing t_i twice cannot reach a marking in P_2 because P_2 is not directly updated by t_i . Firing t_o before t_i is impossible because P_1 is empty initially that cannot enable t_o if t_i has not yet fired. The only feasible firing sequence is $t_i \rightarrow t_o$. We can build a reduced formula ϕ' :

$$\phi' = I(s_0) \wedge (t_i(s_0, s_{temp}) \wedge t_o(s_{temp}, s_2)) \wedge (\neg f(s_0) \vee \neg f(s_1)) \quad (5)$$

where s_{temp} is an intermediate state for a consecutive firings of t_i and t_o , and does not need to be checked.

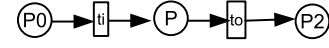


Figure 4: A Simple Model

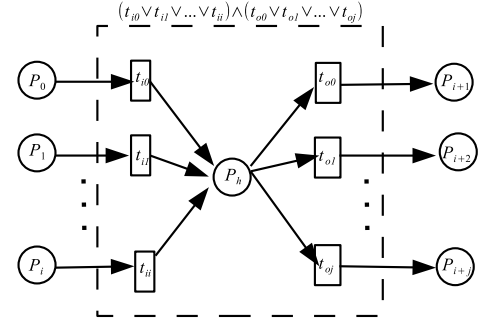


Figure 5: A Structural Pattern

A. A New Structural Pattern

The above observations show that exploring transition dependencies can field much concise formulas that can be solved more efficiently. It is well known that many simple structural reductions such as removing self-loop can be done to a given Petri net to obtain a behavioral equivalent yet simpler Petri nets. However simple net structural transformation rules need to be applied carefully with regard to high level Petri nets since the removed net elements may contain critical information such type information in places, arc label expressions, and constraints associated with transitions, as a result the reduced net may not be behavioral equivalent to the original net. We have developed the following general structural and conceptual (only used during formula translation) reduction rule, and proved its correctness - behavioral preservation.

Figure 5 shows a place P_p that is connected by a set of transitions $T_p = \{t_{i0}, t_{i1}, \dots, t_{iu}, t_{o0}, t_{o1}, \dots, t_{ov}\}$. P_p 's input transition set is $T_{pi} = \{t_{i0}, t_{i1}, \dots, t_{iu}\}$ and output transition set is $T_{po} = \{t_{o0}, t_{o1}, \dots, t_{ov}\}$.

Under the following conditions:

- 1) All the arc label connected to P_p are simple variables;
- 2) P_p is neither an initial marking place nor a property identified place;
- 3) P_p is the only output place of all T_{pi} and the only input place of all T_{po} .

Let s' be a successor state of s and s'' be a successor state of s' . A new and much more concise subformula (equation 6) of ϕ_k is obtained:

$$T_p(s, s'') = (t_{i0}(s, s') \vee t_{i1}(s, s') \vee \dots \vee t_{iu}(s, s')) \wedge (t_{o0}(s', s'') \vee t_{o1}(s', s'') \vee \dots \vee t_{ov}(s', s'')) \quad (6)$$

B. Experiment Results For Refined Transition Formula Construction

Figure 6 shows a share memory model in HLPN. In this model, the pattern can be applied to place $P_{OwnMemAcc}$'s input transition $T_{Begin_Own_Acc}$ and output transition $T_{End_Own_Acc}$, thus the pattern is defined as

SANGE – Stochastic Automata Networks Generator

A tool to efficiently predict events through structured Markovian models

Joaquim Assunção ^{*†}, Paulo Fernandes ^{*}, Lucelene Lopes ^{*}, Angelika Studeny [†], Jean-Marc Vincent [†]

^{*}PUCRS University – Computer Science Department – Porto Alegre, Brazil

[†] Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France

{joaquim.assuncao, angelika.studeny, jean-marc.vincent}@inria.fr

{joaquim.assuncao, paulo.fernandes, lucelene.lopes}@puccrs.br

Abstract

The use of stochastic formalisms, such as Stochastic Automata Networks (SAN), can be very useful for statistical prediction and behavior analysis. Once well fitted, such formalisms can generate probabilities about a target reality. These probabilities can be seen as a statistical approach of knowledge discovery. However, the building process of models for real world problems is time consuming even for experienced modelers. Furthermore, it is often necessary to be a domain specialist to create a model. This work illustrates a new method to automatically learn simple SAN models directly from a data source. This method is encapsulated in a tool called SAN GEnerator (SANGE). This new model fitting method is powerful and relatively easy to use; therefore this can grant access to a much broader community to such powerful modeling formalisms.

1 Introduction

Stochastic Automata Networks (SAN) is a powerful formalism to describe systems as stochastic models. Through these models we can derive probabilities concerning some event or set of events of a system. Our research group has a record of successful development of stochastic models for behavior prediction from several domains, *e.g.*, geological events [2], production lines [6] and distributed software development teams [7]. In all these examples, the model construction required domain specialists and a large amount of stochastic modeling knowledge. The resulting models are very accurate in predicting the behavior of the realities as could be verified by comparison with records of each reality behavior.

Typically, SAN model construction is a top-down driven approach, *i.e.*, first the target reality is analyzed, then its behavior is translated into a stochastic model. Once we have a complete SAN model, it is possible to use a collection of specialized algorithms that can solve it [4]. The problem of this approach is that it is specific to a given system. In other words, each new system must be carefully analyzed before the creation of the model. This analysis usually is performed

via handmade steps such as data analysis and data selection.

In a previous work [1], we proposed a bottom-up process to forecast events using time series and stochastic models (Figure 1). This modeling approach also speeds up the time to develop a representative model from input data. However, we did not have a technique to automatically generate SAN code, but only plain Markov chains (MC). The extension of this previous work to generate SAN models increases our potential to handle more complex (multidimensional) data.

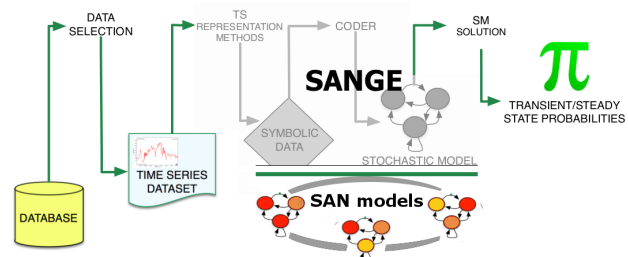


Figure 1: A Dimensionality Reduction Process to Forecast Events Through Stochastic Models [1] enhanced by the generation of SAN models (SANGE).

A bottom-up, MC-based, approach gives a solution for any generic model. However a plain (unstructured) MC is usually a limited, memory expensive model. Limited in the sense that you have a bulk representation for a system, regardless of how complex it may be. Memory expensive because a system with S states is represented by a transition probability matrix of S^2 . In the best case, the number of non-zero entries will be in the order of $2S$.

SAN formalism is modular and its model representation is composed by a collection of sub-systems, which are usually much more compact benefiting from tensor representations. Thus, in this paper we show a new approach to fit SAN models directly to data, reducing the model size. This approach has been implemented in a tool called SANGE (SAN

DOI reference number: 10.18293/SEKE2015-087

Generator) that automatically generates SAN models from a dataset.

The excessive human effort to construct models can be avoided with a tool that handles the formal tasks to convert data into state transitions. Consequently, the user can focus on more interesting tasks, such as interpreting the results and applying the gain knowledge. Additionally, SANGE performs dimensionality reduction using time series representation methods [9]. Thus, SANGE is also capable of automatically fitting the model to input data, possibly achieving better models than those made by humans.

Our algorithm was inspired by a well known and broadly used formalism, Hidden Markovian Models (HMM), and its fitting algorithm (Baum-Welch, BW) [3]. Thus, the HMM user only needs to understand the modeling basics and how to interpret the generated HMM model.

The BW algorithm is a special case of the Expectation-Maximization algorithm, using forward-backward probabilities to estimate the model parameters. Our approach can be seen as an adaptation of this algorithm implementing only the forward procedures. However, our solution works with a structured formalism, which naturally can provide higher accuracy and can be more flexible and user-friendly to describe a system.

2 Computational Kernel

SANGE's main objective is to reduce the time spent on generating SAN models, thus, opening the use of SAN models to non-specialists. However, our solution needs records of a system behavior in the form of time series that will be assigned to the variables of interest for the system.

SANGE's basic operation consists in the composition of a set of time series describing how system variables behave [8]. Considering a system with n interest variables ($v^{(i)}$ with $i = 1, \dots, n$.) we need a behavior sample of the system in the form of n time series with the successive values of the variables through time. With these time series, the first step is to identify the points of interest in time as the time ticks where at least one of the variables changes its value. Once the time ticks of interest are identified, the second step is to determine the possible values for the variables in order to define the stochastic model. This process is summarized in an example with three time series in Figure 2 and Figure 3 showing the identification of 11 time ticks (a) and the three succession of values for variables $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$.

The examples of Figure 2 and Figure 3 results in three automata where local states are given by the observed values for each variable. Transitions events refer to the possible changes in states. These can happen locally within one automaton or simultaneously for several automata (synchronizing events). Figure 4 presents the SAN model for this example.

To compute the rates of local events we must count how

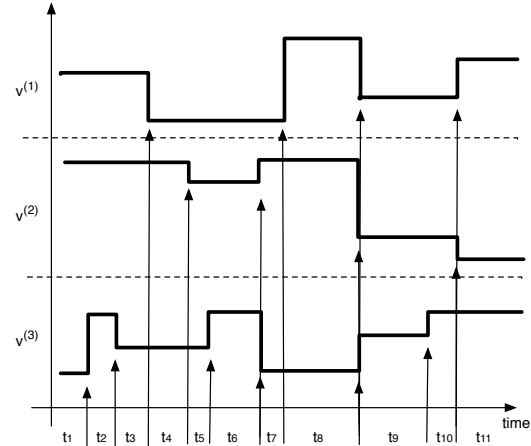


Figure 2: Example of SANGE basic process to three time series - identifying time ticks of interest.

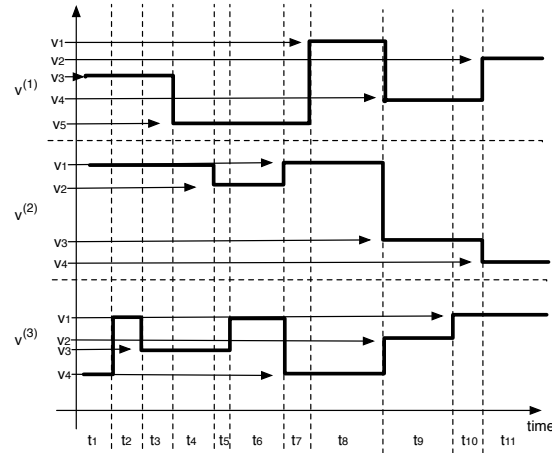


Figure 3: Example of SANGE basic process to three time series - identifying transitions between variable states.

many transitions take place starting in each local state. For example, consider the transition from state $v_3^{(1)}$ to state $v_5^{(1)}$. Observing all time ticks, we see state $v_3^{(1)}$ at the end of ticks t_1 , t_2 and t_3 . In t_1 and t_2 automaton $V^{(1)}$ does not change state, before going to $v_5^{(1)}$ in t_3 . Therefore, one in three times the event $v_3^{(1)} \rightarrow v_5^{(1)}$ occurs, and the rate of this event is $1/3$.

For synchronizing events the computation is similar, but since more than one automaton is concerned, we now have to count the number of times that a certain number of combination of states occurs. For example, we look at automata $V^{(1)}$ and $V^{(2)}$ and the combination of states $v_4^{(1)}$ and $v_3^{(2)}$. This combination occurs at the end of time ticks t_9 and t_{10} , and the synchronizing event happens at one of these two time points (after t_{10}), hence, its rate is $1/2$.

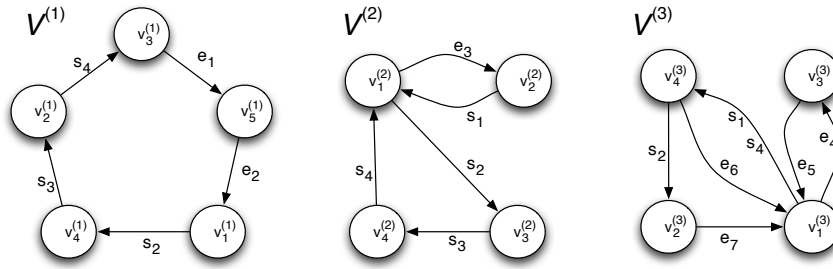


Figure 4: Equivalent SAN model for the three time series example of Figure 3.

From a practical point of view the current version of SANGE can generate either a SAN model or a Markov chain output (which is basically a SAN model with a single automaton and only local events). The output is in the format of a .san file, that can be directly fed into a SAN solver, such as PEPS [4] that performs state of art Kronecker solutions [5].

The presented example consists of quite short time series, which limits the validity of the generated model. In real cases, much longer time series, where events are represented by a large number of state successions, must be considered in order to see statistically relevant patterns.¹

3 Example: Weather in Gotham city

To facilitate the understanding, we use a classical example for Markov models, *i.e.*, forecasting weather events [10]. The most basic example is a Markov chain with 3 states, representing **R**aining, **S**unny and **C**loudy (Figure 5). Probabilities are assigned to the transitions between states as well as staying in the same state.

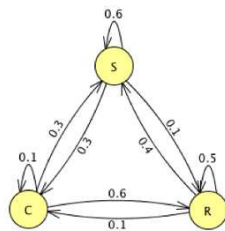


Figure 5: Classical example Markov chain model.

By solving this model we can achieve the transient and stationary probabilities of being in one of the three states. As the only representation of such a model is by a probability matrix, it can be computationally hard to handle for real world applications with many states. Furthermore, the best known formalisms, Markov chains and HMMs, can not integrate models with multiple automata in a unique system,

¹Here we limited the amount of data for the sake of clarity, yet an extended version of this work is available in <https://hal.inria.fr/hal-01149604>

i.e. they do not have a structure for this. By using SAN with SANGE it is possible to generate a structured version, which allows us to assemble more elements to our model and solve those as one.

SANGE encapsulates the techniques described in the Section 2; thus, it provides an interface for this basic and more advanced statistical tools. As pointed out before, the algorithm merges the SAN and TS characteristics.

The following example does consider nor real data neither the adequacy of the model; Our goal here, is to illustrate how SANGE works with multiple variables and how easy is to create a model with it. Although SANGE is a prototype, the basic functions are implemented and some basic models can be generated.

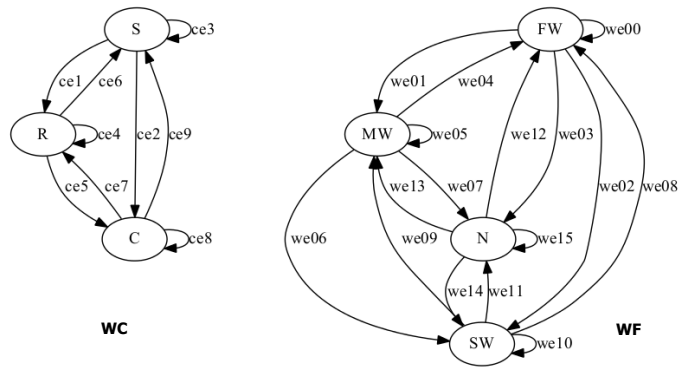


Figure 6: Generic SAN model for weather conditions and wind force. “**ce**” means climate event and “**we**” wind event.

For ease of understanding, Figure 6 shows a generic model with 2 automata, WC for weather conditions and WF for wind force. The states’s labels WC are **R**aining, **S**unny and **C**loudy. In WF, the states correspond to the wind velocity, **F**ast, **M**edium, **S**low and **N**one.

Assuming that this model is accurate to predict the wind and the climate of a city called *Gothan*, we want to know the probability of *Gothan* facing rain and fast wind at the same time. We do not have many records, so we need to learn this probability by a small sample which is formatted as Table 1.

In this case, the probability to have rain and fast wind is

Raw data		Symbolic data	
weather	wind speed	weather	wind speed
cloudy	48	b	d
raining	16	c	b
sunny	26	a	c
...
sunny	9	a	a

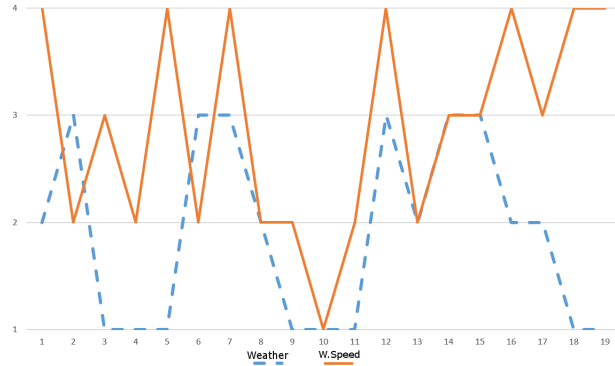


Figure 7: Line plot after the symbolic representation. Each letter is assigned to a value, $a = 1$, $b = 2$, $c = 3$, $d = 4$.

4.5%. In this example a basic combination of two automata was used with a maximum number of 80 states. However, the number of automata and states could be much larger. This is a very compact representation and through SANGE; it scales better and demands less effort than the equivalent Markov model, allowing an easier derivation of probabilities in large models.

4 Final Remarks

As the core technique to data mining, statistics are important to knowledge discovery. It allows us to infer probabilities through samples instead of considering the complete behavior data. Stochastic formalisms are heavily based on probabilistic techniques. The use of stochastic modeling tools is promising to forecast the behavior of systems which can be described as sets of time series.

Our main achievement was to introduce SAN formalism to the process (illustrated in Figure 1) in an automated way, allowing non-specialist users to take advantage of SAN's structure and solutions. Through the automatic fitting, we create a bottom-up approach that can be broadly used, once that the learning process avoids the human effort to manually create such models. Compared to the traditional modeling approach, our implemented solution, SANGE, represents an interesting option that simplifies the effort to construct SAN models. As SANGE is a first attempt to automatically

generate SAN models that can be useful to real world datasets, we have introduced a new method for knowledge discovery through stochastic modeling.

For future work, we will improve our algorithm by adapting the forward-backward procedures from BW algorithm, improving SANGE capacity to handle more complex SAN models for real datasets.

5 Acknowledgments

"This work was conducted during a scholarship supported by the International Cooperation Program CAPES/COFECUB at the University Joseph Fourier. Financed by CAPES - Brazilian Federal Agency for Support and Evaluation of Graduate Education within the Ministry of Education of Brazil."

References

- [1] J. ASSUNÇÃO, P. FERNANDES, L. LOPES, AND S. NORMEY, *A dimensionality reduction process to forecast events through stochastic models*, in SEKE 2014, Jul 2014, pp. 534–539. ISBN-13: 978-1-891706-35-7.
- [2] J. ASSUNÇÃO, L. ESPINDOLA, P. FERNANDES, M. PIVEL, AND A. SALES, *A structured stochastic model for prediction of geological stratal stacking patterns*, *Electronic Notes in Theoretical Computer Science*, 296 (2013), pp. 27 – 42.
- [3] L. E. BAUM, T. PETRIE, G. SOULES, AND N. WEISS, *A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains*, *The Annals of Mathematical Statistics*, 41 (1970), pp. 164–171.
- [4] L. BRENNER, P. FERNANDES, B. PLATEAU, AND I. SBEITY, *PEPS2007 - Stochastic Automata Networks Software Tool*, in Proceedings QEST, 2007, pp. 163–164.
- [5] R. M. CZEKSTER, P. FERNANDES, AND T. WEBBER, *Efficient vector-descriptor product exploiting time-memory trade-offs*, *ACM SIGMETRICS Performance Evaluation Review*, 39 (2011), pp. 2–9. doi: 10.1145/2160803.2160805.
- [6] P. FERNANDES, M. O'KELLY, C. PAPADOPOULOS, AND A. SALES, *Analysis of exponential reliable production lines using kronecker descriptors*, *Int. Journal of Production Research*, 51 (2013), pp. 2511–2528.
- [7] P. FERNANDES, A. SALES, A. R. SANTOS, AND T. WEBBER, *Performance evaluation of software development teams: a practical case study*, *Electronic Notes in Theoretical Computer Science*, 275 (2011), pp. 73 – 92.
- [8] J. LIN, E. KEOGH, S. LONARDI, AND B. CHIU, *A symbolic representation of time series, with implications for streaming algorithms*, in Proceedings of the 8th ACM SIGMOD, DMKD '03, New York, NY, USA, 2003, ACM, pp. 2–11.
- [9] J. LIN, E. KEOGH, L. WEI, AND S. LONARDI, *Experiencing sax: a novel symbolic representation of time series*, *Data Mining and Knowledge Discovery*, 15 (2007), pp. 107–144.
- [10] W. J. STEWART, *Probability, Markov Chains, Queues, and Simulation*, Princeton University Press, USA, 2009.

Modeling and Analyzing Adaptive Energy Consumption for Service Composition

Guisheng Fan^{1,2}, Huiqun Yu¹, Liqiong Chen³

¹Department of Computer Science and Engineering

East China University of Science and Technology, Shanghai 200237, China

²Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China

³ Department of Computer Science and Information Engineering

Shanghai Institute of Technology, Shanghai 200235, China

Corresponding author: lqchen@sit.edu.cn

Abstract—In this paper, Petri nets are used to model the different components of service composition, and form the energy consumption model of service composition based on the relationship between components, Agent is also introduced in the energy consumption management process. Then, an adaptive energy consumption strategy are proposed to dynamically ensure that service composition can get the lowest energy consumption. The operational semantics and related theories of Petri nets help establish the correctness of our proposed method. We have also performed two simulations to evaluate our proposed approach. Results show that it can help reveal the structural and behavioral characteristics of energy consumption in service composition.

Index Terms—Service composition, Agent, Petri nets, energy consumption

I. INTRODUCTION

Service composition provides a mechanism for distributed software integration, which primarily concerns the requests of users that cannot be satisfied by any available service[1]. This increased usage of service composition, together with the increasing energy costs[2]. The energy management is highly complex. First, energy consumption is inherently complicated due to the scale, heterogeneity, and concurrent user services that share a common set of service. Second, the operating environment of service composition is dynamic, it is difficult to plan energy consumption schema at design time. When service composition starts to execute, composition can not achieve the goal because of wrong design of composition or can not meet the required energy consumption. The failure will result in a waste of computing resources and increase energy consumption. Therefore, it requires that energy consumption management of service composition must have a certain adaptive ability, which means that service composition can dynamically select the service, in order to complete its execution successfully and meet the required energy consumption.

Contributions. This paper investigates how to model and analyze the energy consumption of service composition based on user requirements. Below summarizes our main contributions: (1) Petri nets[3] are used to model different components of service composition. Agent[4] is introduced in the energy consumption management process. (2) We evaluate the energy consumption of service composition from the operational level(in addition to service, component, connector and Agent). Based on the model, we propose an adaptive energy consumption strategy to dynamically ensure that service composition can get the lowest energy consumption. (3) The operational

semantics and related theories of Petri nets help establish the effectiveness of our proposed method.

The rest of this paper is organized as follows. Section II presents our energy consumption evaluation, and Section III describes how we model the different components of service composition. Next, we show how to analyze the constructed model(Section IV), and then evaluate the proposed method via simulation in Section V. Finally, Section VI surveys related work, and Section VII concludes.

II. ENERGY CONSUMPTION EVALUATION OF SERVICE COMPOSITION

A. Requirements of service composition

As the function of service composition is composed of a number of independent tasks according to a certain composition rules. Each task has a number of available services. The energy consumption of service under each state can be got by testing and simulation. The interaction between components, services, Agents, and subsystems is defined as the connector, and the connector is viewed as the basic element of service composition, so the connector can be seen as service. The energy consumption of connector is considered.

Definition 1: The energy consumption requirement of service composition is a 7-tuple: $\Xi = (WS, AG, AL, RE, TW, RA, OP)$:

(1) The set of service $WS = \{WS_1, WS_2, \dots, WS_n\}$, $WS = (type, op, ra, ep, ei)$, $type, op$ are the type and operations of service, $ep(op) = (etp, enp)$ is the execution time and per unit of energy consumption of the operation. $ei = (ty, func)$ is the set of service interface.

(2) The connector $AL = \{al_1, al_2, \dots, al_n\}$, $al = (data, oa, nl, li)$, $data$ is the transmission content, oa is the associated service or Agent, nl is the energy consumption, li is used to describe the input and output interface.

(3) The set of Agent $AG = \{ag_1, ag_2, \dots, ag_n\}$, $ag = (eg, lg, om, op, gi)$, $eg \subseteq WS$, $g \subseteq AL$ is the set of service and connector of ag ; $om : AG \times RA \rightarrow \{Start, Sleep, reV, raV\}$ is the possible operations that Agent can do for service, op is the set of operation of Agent, gi is the set of interface of ag , ag may have several input and output interfaces.

(4) The set of component $C = \{C_1, C_2, \dots, C_n\}$, $C = (et, tp, rl)$, $et : C \rightarrow WS^*$ is the set of available service of C_i , $tp : C \rightarrow op^*$ is the set of operations that need be realized, rl is a relation function between the components, the main

relationship is sequence(>), choice(+), parallel relationship (||). $VW(W S_i, C_j) = \{C_k | W S_i \in et(C_k) \cup C_k \in C_j\}$ is the set of operation of $W S_i$ for component C_j .

In service composition, the service will realize its function by executing a series of operations. Because the operation of service may be different, which will make the energy consumption of service be different too. Therefore, the energy consumption analysis based on service level may cause large errors. This paper will describe the energy consumption of service composition based on the operation level.

B. Energy consumption evaluation

Because the component in service composition may invoke different services to realize its function, which will make the reachable states of energy consumption model be different. Let C_i invoke $OP(W S_j, C_i)$ of $W S_j$ to realize its function, we will evaluate the energy consumption in the following.

(1)Energy consumption of component C_i

The energy consumption of the operation $op_f \in OP(W S_j, C_i)$ of component C_i is $en(op_f) = etp(op_f) \times etp(op_f)$.

The energy consumption of component C_i is:

$$EN(C_i) = \sum_{op_f \in op(ae_j, tk_i)} en(op_f) \quad (1)$$

(2)Energy consumption of service

The energy consumption of service is:

$$EN(W S_j) = \sum_{C_i \in C(ae_j)} EN(C_i) \quad (2)$$

The average energy consumption that service using to realize the function of component is:

$$avg_EN(W S_j) = \frac{EN(W S_j)}{|C(W S_j)|} \quad (3)$$

(3)Energy consumption of Agent and service composition

The energy consumption of Agent is composed by service and connector, so the energy consumption of Agent is:

$$EN(ag_j) = \sum_{W S_j \in W S_{ac} \cap W S(ag_i)} EN(ae_j) + \sum_{la_j \in AE_{ac} \cap lg(ag_i)} en(la_j) \quad (4)$$

The energy consumption of service composition is:

$$EN = \sum_{ag_i \in AG} nl(ag_i) = \sum_{ae_j \in AE_{ac}} EN(ae_j) + \sum_{la_j \in AE_{ac}} en(la_j) \quad (5)$$

We can quantitatively evaluate the energy consumption of service composition by using the above formula.

III. MODELING SERVICE COMPOSITION

A. Modeling service composition

In this section, we will use Petri nets to model different components of service composition, then construct the energy consumption model of service composition based on its execution process. In addition, we mark the service, Agent, connector in the front of place and transition.

a) *Modeling service:* The model of service is modeled as following, $p_i^I, p_{ie}^I, p_o^O, p_{dt}^O$ are used to describe the startup, suspending, running and overtime interface, t_i, t_e and t_{ie} are used to describe the startup, finished and suspending operation. The execution of op_i in service is: $t_{a,i}$ is used to describe the execution of the operation, $ct(p_{a,i}) = etp(op_i)$ is the execution time of operation, $en(t_{a,i}) = enp(op_i)$. If the service is in the overtime or suspended state, then invoke the transition t_p to make the operation be in the interrupted position $p_{p,i}$. If the transition can re-execute ($p_{r,i}$), then invoke transition $t_{r,i}$ to make the operation be in the available position ($p_{w,i}$).

b) *Modeling component:* The model of component is shown in Fig.1: First, we introduce p_w to store the set of available service, $\forall W S_j \in W S$, there is $d_j^w \in M_0(p_w)$. If C_i gets the input parameter $p_{s,i}$, then invoke the transition $t_{pp,i}$ to allocate the appropriate service for C_i , and the component will be in the waiting for execution(P_{wa}^i). If C_i gets the input parameter $p_{s,i}$, then no available service can realize the function of component C_i and $t_{f,a,i}$ is used to output the fault.

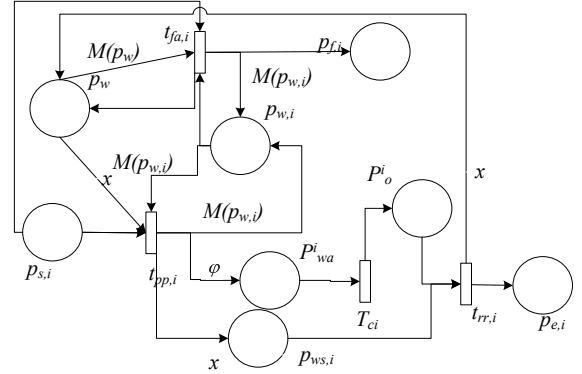


Fig. 1. The model of component

c) *Modeling connector:* The model of connector is shown in Fig.2, let two basic elements of connector be A and B. Then the system will introduce two interfaces to send and receive the info from A, B.

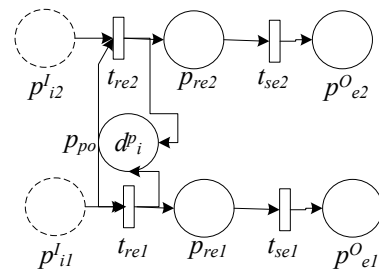


Fig. 2. The model of connector

d) *Modeling Agent:* The model contains the execution of component and connector in Agent, that is, the execution environment of Agent. Agent will get the information from the environment and compute the startup service based on the adaptive energy consumption strategy. t_{st} is used to start the service, while t_{ns} is used to make the remaining service be in the suspended state. t_e represents the termination of Agent.

The service will do the related adjustment when it receives the instruction of Agent.

The steps for constructing energy consumption model are. Constructing the model of services based on their attributes. Introducing t_{st} and p_{st} to describe the beginning operation and position. t_{en} and p_{en} are used to describe the termination operation and position of whole application. Computing the set of startup service by using the adaptive strategy, and initializing the place in the model. Setting initial marking $M_0(p_s) = \varphi$, while setting the priority of transition t_{dt} and transitions in the system level be 4. In this paper, we divide the priority of transition into 5 level, which can be adjusted according to the actual requirements.

IV. ADAPTIVE ENERGY CONSUMPTION STRATEGY AND ANALYSIS

In this section, we will propose an adaptive energy consumption strategy for service composition based on Agent. Then dynamically compute the set of startup service based on the state and attributes of the current service.

A. Adaptive energy consumption strategy

A series of operations that service ws is used to realize the function of C_i are called the path of ws . Let the path that WS_i uses to realize the function of component C_j be $Lat(WS_i, C_j) = \{Lat_1, Lat_2, \dots, Lat_n\}$, $LaC = \{op_{k,1}, op_{k,2}, \dots, op_{k,f}\}$. The energy consumption of path Lat_i is:

$$En(Lac) = \sum_{ap_{k,l} \in Lat_k} enp(op_{k,l}) \times etp(op_{k,l}) \quad (6)$$

The average energy consumption that service WS_i uses to realize the function of component C_j is:

$$Avg_en(wsi, C_j) = \frac{\sum_{Lat_k \in Lat(aei, TK_j)} en(Lat_k)}{|Lat(aei, TK_j)|} \quad (7)$$

B. Analysis technique

In this section, we will verify the correctness of proposed method based on the reachable states of energy consumption model and the operation semantics of Petri nets.

Theorem 1. *Let $R(\Omega)$ be the reachable state of energy consumption model Ω , $\forall S \in R(\Omega), \forall WS_i \in WS$, the set of startup service is WS_{ac} , then:*

(1) *If $WS_i \in WS_{ac}$, $\exists S \in R(\Omega)$, which makes $|M(WS_i \bullet p_e)| = 1$.*

(2) *If $WS_i \notin WS_{ac}$, $\exists S \in R(\Omega)$, which makes $|M(WS_i \bullet p_p)| = 1$.*

Proof: If $WS_i \in WS_{ac}$, we may set $WS_i \in eg(ag_j)$, that is, WS_i belongs to the corresponding Agent. We can set $ag_j \bullet P_i^i \in ag_j \bullet t_{st}^i$ in the modeling process of ag_j , therefore, $S_1 \in R(S_0)$, which makes $M_1(ag_j \bullet p_i^i) \neq \emptyset$ and $M_1(ag_j \bullet p_{ie}^i) = \emptyset$, because $M_0(ag_j \bullet p_{st}) = (d_j^a, 1)$ and $ag_j \bullet p_{st}^i = ag_j \bullet t_{st}$, therefore, $M_1(ag_j \bullet p_{st}) = (d_j^a, 1)$. Form the modeling process of service, we can get that transition $WS_j \bullet t_i$ can be fired, while $WS_j \bullet t_{ie}$ isn't enable. Therefore, $\exists S \in R(\Omega)$, which makes $|M(WS_i \bullet p_e)| = 1$. Similarly, we can

prove (2) establishes from the modeling process that Agent does the suspended operation for service. ■

Theorem 1 explains that the energy consumption model can correctly describe the interaction between the Agent and service, such as, the system get the info of service, Agent startups and suspends the service.

Theorem 2. *Let $R(\Omega)$ be the reachable state of energy consumption model Ω , $\forall S \in R(\Omega), \forall WS_i \in WS$, then:*

(1) *If $|M(WS_i \bullet p_p)| = 1$, $\exists S' \in R(\Omega)$, which makes $|M'(WS_i \bullet p_e)| = 1$.*

(2) *If $|M(WS_i \bullet p_{p,j})| = 1$, $\exists S' \in R(\Omega)$, which makes $|M'(WS_i \bullet p_{g,j})| = 1$.*

Proof: $\forall WS_i \in WS$, because $|M(WS_i \bullet p_p)| = 1$, therefore, ws is in the suspended state under S . Let WS_i restart after a period of time, We will prove the suspend of WS_i will not affect the restart of it, we can prove it from two two aspects: First, we will prove WS_i can be in the running position again. Because $WS_i \bullet p_i^i = \{WS_i \bullet t_i, WS_i \bullet t_r\}$, $\bullet WS_i \bullet t_r = \{WS_i \bullet p_i^i, WS_i \bullet p_p\}$, so $WS_i \bullet t_r$ has the right to fire and the priority of $WS_i \bullet t_r$ is 0. Therefore, the firing of $WS_i \bullet t_r$ under S is effective, we may set that S_1 will reach S_2 by firing $WS_i \bullet t_r$, then there is $|M_2(WS_i \bullet p_w)| = |M_1(WS_i \bullet p_i^i)| = 1$, because $\bullet WS_i \bullet t_i = \{WS_i \bullet p_i^i, WS_i \bullet p_w\}$, therefore, the firing of $WS_i \bullet t_i$ under S_2 is effective. Second, we can prove that all operations can be executed properly, $\forall op_f \in op(WS_i)$ may be in $p_{a,f}$ or $p_{e,f}$ when WS_i is suspended. Therefore, $\exists S' \in R(S)$ which makes $|M'(WS_i \bullet p_e)| = 1$.

We can prove the sub-proposition(2) in the similar way. ■

Theorem 2 illustrates that the service and its operations can execut properly when Agent needs to restart the suspended service, thus realizing its function.

V. EXAMPLES

In order to evaluate the effectiveness of proposed method, we will use experimental platform Windows 7, C# language to implement a prototype for analyzing the constructed model. First, we will generate available services as service resource. Each service at least has the basic information such as energy consumption, the function and the set of operation, etc.

Experiment 1. The goal of this experiment is to analyze the effectiveness of adaptive strategy, the specific steps are:

(1) Taking 400 services, 50 components, 10 Agents and dividing them into 10 groups, each group corresponds to a Ligo system, then do Step (2)-(3) to each Ligo system.

(2) Selecting 10 groups of service, then compute the energy consumption of each Ligo system[5] based on each set of available service;

(3) Using adaptive strategy to compute the set of startup service of service composition, then compute the whole energy consumption of each system based on the set of startup service.

The results of Experiment 1 are shown in Fig.3, we can get: (1) the energy composition of the set of service will change when the attributes of service has changed. (2) the overall energy consumption of service composition can be reduced by using the adaptive strategy, the highest reduction is 34.5% , and the lowest reduction is 3 %.

Experiment 2. Experiment 2 analyzes the relationship between the state space and available service, the steps are:

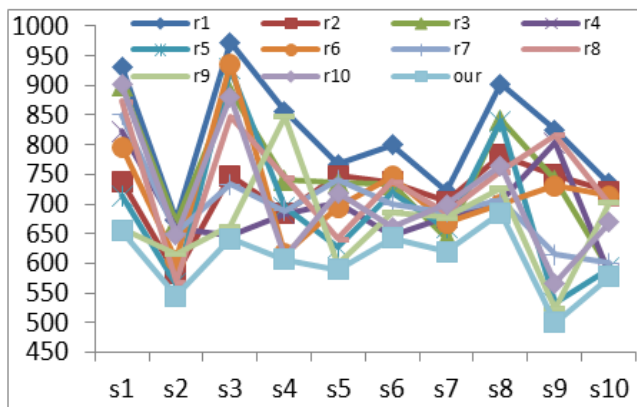


Fig. 3. Experimental results of Experiment 1

(1) Taking 20 services, 5 components, 1 Agent to construct the resource of Ligo system;

(2) Adding the service to system by 10 each time, that is, 30, 40, 50, 60, 70, 80, 90, then compute the set of startup service and the number of reachable states of model.

From the results of Experiment 2, we can get that: the state space of energy consumption model is 32 in all cases, that is, the increase of available service will not effect the state space of energy consumption model. The state of model will not increase with the available service increasing. The proposed method can be used to model and analyze the energy consumption of large scale service composition.

VI. RELATED WORKS

There is a vast amount of research available on adaptive designs for different areas. The work in [6] presents a system architecture to monitor, interpret and analyze system events in order to implement self-healing and self-adaptive systems. The architecture presented in [7] is focused on service level agreement management in a Service Oriented Architecture. The approaches defined in the above don't consider the different components of service composition.

Agent system design has emerged as a powerful approach to perform tasks or solve problems in a decentralized environment. A framework for building an adaptive Learning Management System has been proposed in [8]. Reference [9] introduces an Agent-oriented Model-Driven Architecture. Agents use hierarchical business knowledge models with business process rules at the top, business rules to control policy and logic in the middle and a base layer defining business concepts. Later, the authors design a flexible method that supports a range of coordinator components[10]. However, the approaches defined in the above don't consider the operation of service, and they don't involve in how to filter the service.

Many research efforts for service oriented computing have adopted formal methods techniques to leverage its mathematically precise foundation for providing theoretically sound and correct formalisms. A novel method for runtime monitoring of composite services is proposed by [11], they employ process algebra as the primary formalism to express specifications. A similar approach is given by [12], the authors present a CSP-based workflow framework for intelligently navigating service composition. We have proposed an approach to constructing

the reliable service composition[13]. Almost all of the aforementioned formalisms cover basic and structured activities of service composition, but they are unable to ensure that the constructed model can meet the users' specific requirements, such as energy consumption. Meanwhile, the approaches defined in the above can not solve the problem of dynamically selecting available service.

VII. CONCLUSION

In this paper, we proposed new solutions for optimizing energy consumption for service compositions. The special features of the proposed model include: (1) Petri nets are used to describe different components of service composition. Agent is introduced to the energy consumption management process of service composition. (2) The adaptive energy strategy is proposed, which is used to dynamically select the available service that meets the requirement for component, thus reducing the energy consumption of service composition. (3) The operational semantics and related theories of Petri nets help prove the correctness of proposed method. Finally, we conduct several experiments to evaluate the effectiveness of the proposed method.

ACKNOWLEDGMENT

The work is partially supported by the NSF of China under grants No. 61173048 and 61300041. Research Fund for the Doctoral Program of Higher Education of China under Grants No. 20130074110015. The Fundamental Research Funds for the Central Universities under Grant No.WH1314038.

REFERENCES

- [1] S. Marston, Z. Lia, S. Bandyopadhyaya, et al. Cloud computing—the business perspective. *Decision Support Systems*. 2011, 51(1): 176-189.
- [2] C. Wang, M. de Groot, P. Marendy. A Service-Oriented system for optimizing residential energy use. *IEEE International Conference on Web Services(ICWS 2009)*, IEEE Computer Society, Washington, DC, USA, 2009:735-742.
- [3] M. TADAO. Petri nets: properties, analysis and application. *Proceedings of the IEEE*. 1989, 77(4):540-581.
- [4] C. Antonio, C. Massimo, G. Salvatore, V. Seidita. Agent-Oriented software patterns for rapid and affordable robot programming. *Journal of Systems and Software*, 2010,83(4):557-573.
- [5] G. Juve, A. Chervenak, E. Deelman, et al. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 2013, 29(3): 682-692.
- [6] I. Al-oqily, B. Subaih, S. Bani-Mohammad, et al. A survey for self-healing architectures and algorithms. *Proceedings of the 9th International Multi-Conference on Systems, Signals and Devices (SSD)*, IEEE Computer Society, Washington, DC, USA, 2012: 1-5.
- [7] R. R. Aschoff, A. Zisman. Proactive adaptation of service composition. In: *processing of the 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, IEEE Computer Society, Washington, DC, USA, 2012: 1-10.
- [8] Y. Mahkameh, B. Ardeshir. A context-aware adaptive learning system using agents. *Expert Systems with Applications*, 2011,38(4): 3280-3286.
- [9] X. Liang, G. Des. Agent model: software adaptivity using an agent-oriented model-driven architecture. *Information and Software Technology*, 2009,51(1):109-137.
- [10] J. Lian, S. Shatz, X. He. Flexible coordinator design for modeling resource sharing in multi-agent systems. *Journal of Systems and Software*, 2009,82(10):1709-1729.
- [11] K. Mohsen, J. Saeed. WSCMon: Runtime monitoring of web service orchestration based on refinement checking. *Service Oriented Computing and Applications*. 2012,6(1):33-49.
- [12] X. Song, W. Dou, J. Chen. A workflow framework for intelligent service composition. *Future Generation Computer Systems*. 2011, 27(5): 627-636.
- [13] G. Fan, H. Yu, L.Chen, D.Liu. Petri net based techniques for constructing reliable service composition. *Journal of Systems and Software*, 2013, 86(4): 1089-1106.

Modeling and Analyzing Publish-Subscribe Architecture using Petri Nets

Junhua Ding^{1,2}

1) Dept. of Computer Science
East Carolina University
Greenville, NC 27587
dingj@ecu.edu

Dongmei Zhang²

2) School of Computer Sciences
China University of Geosciences
Wuhan, Hubei, China
jjielee@163.com

Abstract — Software architecture is the foundation for the development of software systems. Its correctness is important to the quality of the software systems that have been developed based on it. Formally modeling and analyzing software architecture is an effective way to ensure the correctness of software architecture. However, how to effectively verify software architecture and use the results from formal modeling and analysis is important to the application of the approach. In this paper, software architecture is modelled using high level Petri nets, and the model is then checked with a model based testing tool called MISTA, and bounded model checking tool Alloy to ensure the correctness of the model. The approach is designed as a two-phase process consisting of model-based testing and bounded model checking to ensure it is both practical and rigorous for analyzing software architecture. We illustrated the idea and procedure via modeling and analyzing the Publish-Subscribe architecture. The result has shown that combining bounded model checking with model based testing is an effective extension to ensure the development quality.

Keywords- software architecture; Petri net; model checking; model based software testing; publish-subscribe architecture

I. INTRODUCTION

Software architecture is an overall structure of a software system, which consists of a group of components and the connections among components in addition to the constraints applying to the connections. It is the foundation of product lines and many software systems were developed based on it. Therefore, correctness of software architecture is important to the quality of software systems that have been built on it. Formal modeling and analysis of software architecture offers a rigorous way to ensure the correctness of software architecture, which has been discussed in many articles [5]. However, results of formal modeling and analysis are difficult to be directly used for analyzing software implementation that was built based on the formal models due to the specification gap between models and their implementations. For example, if a model is specified using Petri nets, and the implementation language is Java, then the model checking results (*e.g.*, counter examples) of the Petri nets model cannot be directly used for testing the Java program. But model checking a complex Java application is infeasible and testing is still the practical way for program verification. Model-based software testing is an approach to bridge the gap between testing of a software model and its implementation, where models are used for guiding the test generation. In some cases, model level tests are first generated, and then they are transformed into program level tests. MISTA [17] is a model based software testing tool,

which models a software system in high level Petri nets, and then the Petri net model is analyzed with simulation and model checking. Model level tests can be automatically generated according to selected test coverage criteria, and then these tests are automatically transformed into program level tests with help from mapping files. The program level tests can be directly used for testing the implementation. However, due to the grand challenge of modeling of a high level Petri net, model checking capability in MISTA is limited. In this paper, we extended MISTA with bounded model checking for analyzing Petri nets. Alloy analyzer is a bounded model checker for analyzing models specifying in Alloy language, which is a formal specification language based on first order relational logic [3][10]. Alloy analyzer is a constraint solver for automatically checking an Alloy model that specifies the structural constraints and behaviors of a software system [3]. Alloy finds all model instances for satisfying a checked property within the bounded scope, and it provides a visualization tool to illustrate all instances. Comparing the graphic instance to the corresponding Petri net model will be useful to better understand the Petri net model and create a better model. In addition, the instances are also useful for creating tests for testing interesting properties in the Petri net.

Publish-Subscribe (pub-sub) architecture is a well adopted event-based software architecture. The pub-sub architecture includes one or more components that publish events, and one or more components that subscribe them. The loose coupling of publish and subscribe components offers the flexibility of updating components and events in a system, but it also brings the complexity of analysis due to the large number of possibility of combination of event transferring scenarios [6]. Several analysis approaches such as model checking [6][8] have been attempted for analyzing pub-sub models. In this paper, we introduce Alloy into MISTA for analyzing Petri nets. First, a Petri net model is modeled and simulated, and then simple properties are verified using MISTA. After that, the Petri net model is converted into an Alloy model, which will be analyzed using Alloy analyzer. The analysis results can be used for improving the Petri net model and guiding generating tests for interesting properties. The analysis process is illustrated through modeling a general model of the pub-sub architecture in Petri nets. The general model can be easily extended for different versions of the pub-sub architecture. Based on the Petri net model, the pub-sub architecture was modelled in Alloy, and analyzed for interesting properties using Alloy analyzer. A Petri net model can be automatically transformed into an Alloy model.

The main contribution of this paper is due to a two-phase rigorous and practically useful approach for analyzing software architecture. Since software architecture is the foundation for the implementation of many software systems, it is important to provide an easy-to-use technique such as simulation and testing for analyzing software architecture when they are still in the early development phase. But simulation and testing is not enough to ensure the correctness of important properties in software architecture. Rigorously checking the architectural model is necessary for ensuring the quality of the architecture especially in the later modeling phase. In our approach, the model-based testing assists ones to understand the modeling of software architecture, to check simple assertions and to test special scenarios for building a correct software architecture. In addition, model checking ensures the correctness of important properties modeled in the architectural model. The bounded model checker Alloy was smoothly extended to model based testing tool MISTA for enhancing the features in MISTA. Modeling and analyzing the pub-sub architecture is used to explain the idea and process, and to show the effectiveness of the proposed approach.

The rest of this paper is organized as follows: Section 2 presents a brief introduction to Alloy, PrT nets, model-based software testing and its tool MISTA. Section 3 introduces the proposed model based testing of the pub-sub architecture in Petri nets using MISTA. Section 4 discusses how to extend Alloy into MISTA, model and analyze the pub-sub architecture using Alloy, and how to convert a Petri net into an Alloy model. Section 5 reviews the related work, and section 6 concludes this paper.

II. BACKGROUND

A. Alloy

Each Alloy model is specified in Alloy language to define how to check the occurrence of a state change [3]. Each model represents a set of model instances, and Alloy analyzer is used to search for instances or counterexamples of a model. Alloy analyzer is a bounded model checker for analyzing a model within a finite scope a user specifies [3]. The analysis is sound and complete within the scope so that it never misses a counterexample within the scope. Alloy analyzer either finds a solution that satisfies a predicate defined in the model, or a counterexample that violates a given assertion [11]. An Alloy model includes a number of signatures and facts. A signature defines a set and a group of atoms associating with the set, and a fact defining a constraint that is assumed always to hold in the model. Analysis of a model is conducted for checking a predicate or an assertion of the model. The details of Alloy analyzer and language can be found in the project website [3] and the book [10]. Fig. 1 is an Alloy model (modified based on the original model described in the tutorial of Alloy [3]) for defining a simple file system, which includes files, directories and root. *sig FSOBJ* defines objects in a file system, *sig File*, *sig Dir* and *sig Root* define files and directories, which are also objects. The three facts define the global constraints that are: each directory is the parent of its contents, each object is either a file or directory, and a root does not have a parent. The *assert* declares that a path is acyclic, and check it for scope of 5 [3].

```

module fileSys
  abstract sig FSOBJ { parent: lone Dir}
  sig Dir extends FSOBJ { contents: set FSOBJ}
  sig File extends FSOBJ { }
  one sig Root extends Dir { } { no parent }
  fact {all d: Dir, o: d.contents | o.parent = d}
  fact { File + Dir = FSOBJ}
  fact { FSOBJ in Root.*contents}
  assert acyclic {no d: Dir | d in d.^contents}
  check acyclic for 5

```

Figure 1. A sample Alloy model

B. PrT Nets

Predicate/Transition (PrT) nets are a high level Petri net for specifying concurrent systems. The definition of PrT nets used in this paper is same as the one defined in [18].

Definition 1 (PrT net) A PrT net is a tuple $(P, T, F, \Sigma, L, \varphi, M_0)$, where: P is a finite set of predicates (first order places), T is a finite set of transitions and F is a flow relation. (P, T, F) forms a directed net. Σ is a structure consisting of sorts of individuals (constants) together with operations and relations. L is a labeling function on arcs. φ is a mapping from a set of inscription formulae to transitions, and M_0 is the initial or current marking.

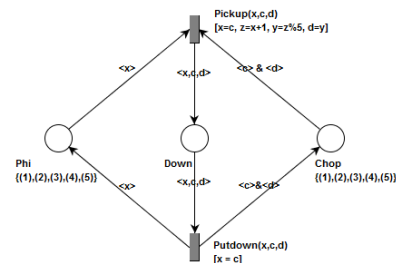


Figure 2. A PrT nets model for dining philosophers

Fig. 2 shows a simplified PrT net model for 5 dining philosophers' problem. The model includes transitions *Pickup*, and *Putdown* represent the action for picking up chopsticks and putting down chopsticks, respectively. The distribution of tokens in places *Phi*, *Chop* and *Down* represents the three states of each philosopher: *thinking*, *full* and *eating*, respectively. Places *Phi* and *Chop* include tokens that are nature numbers representing philosophers or chopsticks, and each token in place *Down* represents a philosopher and his/her two chopsticks. Transition *Pickup* has two input places *Phi* and *Chop*, and one output place *Down*. The guard condition in transition *Pickup* is defined based on the relation between the tokens in place *Phi* and *Chop*: $x=c \ \&\& \ d=(x+1)\%5$, representing that a philosopher must get both of his or her left and right chopstick before he or she can eat (*pickup*) The guard condition in transition *Putdown* is defined based on the relation between the tokens in place *Phi* and *Chop*: $x=c$, representing a philosopher puts down chopsticks at both left side and right side.

C. Model-based Testing and MISTA

MISTA [14][17] is a model-based testing tool for automated generation and execution of tests. It generates tests in model level first and then program level tests are produced through transforming the one at model level. It specifies models in function nets, which is a type of PrT nets extended

with inhibitor arcs and reset arcs [18]. It also provides a language for mapping the elements in function nets to implementation constructs so that it is possible to transform the model level tests into program level tests that can be executed against the system under test. In addition to test generation, MISTA includes simulation and limited model checking functions. It supports the step by step execution and random execution of a function net, and the execution sequences and token changing in each place are visualized for inspection. The test generator generates adequate model level tests (*i.e.*, firing sequences of a function net) according to a selected coverage criterion such as reachability coverage, transition coverage, state coverage, depth coverage, and goal coverage. Test code generator generates test code in a target program language like Java or C++ from a given transition tree [17].

III. MODELING AND TESTING SOFTWARE ARCHITECTURE USING PrT NETS

In this section, we are going to discuss an approach for analyzing a PrT net model using model-based software testing technique. In order to illustrate the basic idea and the process of the two-phase analysis approach, we model and analyze a pub-sub model using MISTA in this section and Alloy in next section.

A. Modeling the Pub-Sub Architecture

The pub-sub architecture is an event-based architecture, which includes one or more publishers that publish contents, and one or more subscribers that consume the contents. The publisher sends its contents as event messages through an event bus, and a subscriber subscribes its contents through an event message classification mechanism that classifies contents as channels [1].

In the PrT net model shown in Fig. 3, a publisher publishes its content as a message *msg* through transition *pub*, and the published content is notified to subscribers via transition *notify*, which models the event bus, and messages are classified as channels and stored in place *channel* by transition *classify*. A subscriber subscribes a channel message via transition *sub*, and the subscribed channel message is sent to the subscriber by transition *classify* when the message is available.

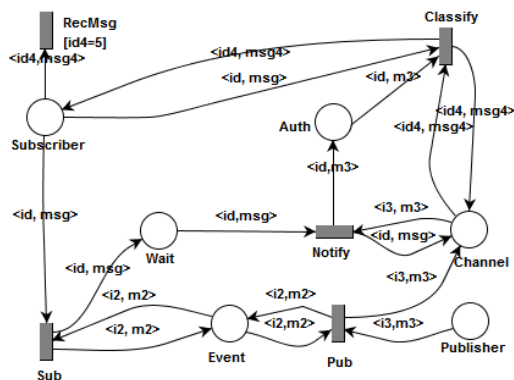


Figure 3. A PrT net model for the pub-sub architecture

B. Testing the Pub-Sub Architecture

As soon as a PrT net model is created and successfully compiled in MISTA, run it with random inputs to help developers to understand the model and detect easily found problems. If the simulation result is acceptable, verification of the goal reachability, assertions and deadlock states is conducted. After that, a set of tests can be generated based on selected coverage criteria, and these tests will be converted into program level tests for testing the corresponding implementation.

First, execute the PrT net model for the pub-sub with valid initial markings to simulate normal running scenarios of the model. For example, check a normal scenario that a publisher publishes a message, which is the type of messages that a subscriber has subscribed. It is important to check that the message is successfully classified and stored in the channel and the subscriber is notified, and finally the message is delivered to the subscriber. An example of the initial marking for checking above scenario in the PrT net in Fig. 3 is:

INIT Event(1, "1"), Publisher(1, "2"), Publisher(1, "1"), Subscriber(11, "0"), Subscriber(11, "s")

Second, verify the reachability of goal states and transitions, assertions and deadlock states in the PrT net for the pub-sub using the model checking capability in MISTA. When the reachability of all transitions of the PrT net in Fig. 3 was checked, transition *RecMsg* was unreachable was found since no any message with *ID=5* was ever sent from any publisher. If a token such as ("5, "2") for place *Publisher* is added to the initial marking, all transitions will be reachable. Given a goal state such as *GOAL Subscriber(5, "2")*, then MISTA will find that the state is reachable. The PrT net model has termination states because published messages are delivered to subscribers and removed from their channels and it is possible that any channel has a message. The model has to be updated if a subscriber only receive copies of its subscribed messages and all messages will stay in the channel for a period of time.

Third, generate adequate tests for selected test coverage criteria using MISTA. For the PrT net model defined in Fig. 3, generate model level tests covering all states, all executable paths, goal states, and others. Since complex scenarios are not feasible to be checked with simulation or the limited model checking in MISTA, these complex scenarios shall be rigorously tested with model level tests thanks to the executable capacity of PrT nets. The model level tests are also used for generating program level tests via mapping the model to its corresponding programs. The program level tests will be used for testing the implementation of the model.

Although the PrT net model was checked with simulation of execution scenarios, verification of important properties, and testing with adequate tests for selected coverage criteria, the model is not guaranteed to be correct. Formal verification of a PrT net model is very difficult, but the analysis process of bounded model checking is fully automatic and it can guarantee the correctness of the checked properties within specified scope. Therefore, bounded model checker Alloy was chosen as an addition to MISTA to further analyze a PrT net model.

IV. ANALYZING SOFTWARE ARCHITECTURE USING ALLOY

In this section, first we discuss how to model and analyze the pub-sub architecture in Alloy, and then we introduce a procedure for transferring a PrT net into an Alloy model.

A. Analysis of the Pub-Sub Architecture

The architecture defined in this section is specified following Acme style [1], which defines software architecture as a group of components and the connection for connecting the components via interfaces in addition to the constraints applying to the connection. The Alloy model of the pub-sub is defined in a hierarchical structure. The basic elements of the general software architecture are defined in an Alloy model serving as the foundation for modeling specific software architecture, and then a model of event based architecture was defined. The foundation model and the model of event based architecture were created based on those models introduced in [11]. The model for the pub-sub architecture was developed through extending above two models. In the foundation model, common software architecture elements such as components, connectors, ports and roles are defined as signatures, and basic constraints of software architecture are defined as facts. The model also defines a group of built-in functions and predicates for checking specific properties or looking for counterexamples. The pub-sub architecture is an event-based architecture that consists of loosely-coupled components that produce and consume events through the ports. In the model of event-based architecture, the signatures include two components: *AnnounceComponent*, and *ReceiveComponent*, for modeling an announce component and a receive component, respectively. Each component includes a set of ports as interfaces; one connector *EventConnector*, models the connector for connecting between roles and ports. Each connector includes two sets of roles as interfaces, two types of roles *AnnoucerRole* and *ReceiverRole*, and two types of ports *AnnoucePort* and *ReceivePort*. The connection between components is implemented through connecting ports to the corresponding roles. The pub-sub architecture is an extension of the event-based architecture with the subscriber selects messages through a message classification mechanism called channel. The following Alloy code is partial of the Alloy model for the pub-sub architecture, and the model was developed based on the Alloy models described in [11]. Channel is defined as a component, and a channel connector is defined for connecting a channel and its subscribers.

```
sig PubPort extends AnnouncePort {}
sig SubPort extends ReceivePort {}
sig PubRole extends AnnoucerRole {}
sig SubRole extends ReceiverRole {}
sig PubComp extends Component {pubPort: PubPort}
sig SubComp extends Component {subPort: SubPort}
sig Channel extends Component {channelPort1:
SubPort, channelPort2: SubPort}
sig EventBusConn extends EventConn {pubRole:
PubRole, subRole: SubRole}
sig ChannelConn extends Connector {subRole:
SubRole, channelRole: ReceiverRole}
```

One fact described as follows is defined to ensure that each component or connector has appropriate number of roles or ports.

```
fact{
(PubComp<:pubPort) in ports
(SubComp<:subscribePort) in ports
(Channel<:channelPort1) in ports
(ChannelConn<:channelRole+ChannelConn<:subRole) in roles
(EventBusConn<:pubRole+EventBusConn<:subRole) in roles
}
```

Based on above model, one can check interesting properties of the model, such as an architectural configuration or a set of constraints. The following predicate defines a set of constraints that define the mapping between ports and roles, and the essential elements in the pub-sub architecture. In the code, *self* is a term defined in Acme as a signature for extending *System*, and *all* and *some* represent universal and existential quantification, respectively.

```
abstract sig System {components: set Component,
connectors: set Connector}
one sig self extends System {}

pred pubsub_constraints() {
some c:self.components|declaresType[c, PubComp]
some k:self.components|declaresType[k, SubComp]
some m:self.components|declaresType[m, Channel]
some n:self.connectors|declaresType[n, EventBusConn]
all self:PubPort |
(all r: self.~attachment|declaresType[r, PubRole])
all self:SubPort |
(all r:self.~attachment|declaresType[r, SubRole])
all self:PubRole |
(#(self.attachment)=1)&&
(all p: self.attachment|declaresType[p, PubPort])
all self:SubRole |
(#(self.attachment)=1)&&(all p: self.attachment|
declaresType[p, SubPort])
all self:EventBusConn |
(some e:self.pubRole|declaresType[e, PubRole] &&
some f:self.subRole|declaresType[f, SubRole])
all self:ChannelConn |
(some e:self.channelRole|declaresType[e, ReceiverRole]
&& some f:self.subRole|declaresType[f, SubRole])
}
```

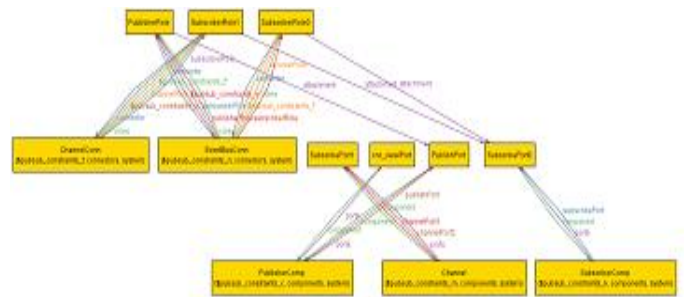


Figure 4. A snapshot of an instance for pub-sub architecture

Checking the predicate with a specified scope such as 5 using Alloy analyzer shall find instances that satisfy the predicate in the pub-sub model. The instances are helpful to understand the architecture and create better PrT net model. Fig. 4 shows a snapshot of an instance. One also can check whether a particular instance of pub-sub can be found or not in the model such as the following configuration of a pub-sub can be found with scope of 4. The configuration defines a simple pub-sub instance which consists of one subscriber, one publisher, one channel, one event bus connection, and one channel connection.

```

pred Config_0(s:SubComp,e:EventBusConn,c:Channel,
             cc:ChannelConn, p:PubComp){
  attached[s.subscribePort, cc.subscriberRole]
  attached[cc.channelRole, c.channelPort2]
  attached[c.channelPort1, e.subscriberRole]
  attached[e.publisherRole, p.publishPort]
}

```

B. Translating a PrT Net into an Alloy Model

The Alloy model discussed in previous section was built directly according to the pub-sub architecture defined in Acme [1] for explaining the analysis process in a better way. In this section, we discuss how to translate a PrT net into an Alloy model to ensure the consistency between a PrT net and its corresponding Alloy model. Since Alloy models are declarative for defining how to recognize something has happened, and PrT net models are operational for defining how something can be accomplished [3]. It is fairly challenge to automate the transformation between a PrT net model and an Alloy model. In this section, we introduce a basic structure of the transformation using the dining philosopher problem defined in Fig. 2 as an example. The idea of the transformation was developed based on the one for translating a regular place transition Petri nets into an Alloy model discussed in [16], which was extended for high-level Petri nets (*i.e.* PrT nets).

First, translate the basic elements (*i.e.* places, transitions, arcs and tokens) and basic constraints in a PrT net into signatures in Alloy.

```

abstract sig Node{flow: set Node}
abstract sig Token{}
abstract sig Place extends Node{tokens:set
Token}{#tokens >= 0}
abstract sig Transition extends Node{inp:set
Place, outp:set Place}
abstract sig Arc{place:Place,tran:Transition}

```

Define common constraints as facts, such as a net consists of places and transitions, and a flow relation is only applied to a place to a transition or a transition to a place:

```

fact {Node = Place + Transition}
fact {all p: Place | p.flow & Place = none}
fact {all t: Transition|t.flow&Transition=none}

```

Since each place has its own type of tokens, it is necessary to define a type of tokens for each place and define each place with its tokens. For example, we defined three types of tokens for the dining philosopher problem.

```

sig Eating extends Token{phi:set Int,left:set
Int,right:set Int}
sig Chop extends Token{left:set Int,right:set Int}
sig Ph extends Token{phi: set Int}

```

And then each place in the PrT net is defined as a signature with its tokens and each transition in the PrT net is also defined as a signature. The constraints are defined by the number of outward flows of each place or transition in the PrT net:

```

sig phP extends Place{token: Ph}{#flow = 1}
sig chopP extends Place{token:Chop}{#flow=1}
sig eatP extends Place{token:Eating}{#flow= 1}
sig Pick extends Transition {}{#flow = 1}

```

```
sig Down extends Transition {}{#flow = 2}
```

Now, the structure of the net is defined as facts in Alloy based on the flow relations in the PrT net. The following Alloy code defines the connections between places and transitions of the PrT net in Fig. 2.

```

fact {all p: phP|one t: Pick|t in p.flow}
fact {all t: Pick|one p: phP|p in t.~(flow)}
fact {all p: phP|one t: Down|t in p.flow}
fact {all t: Down|one p: phP|p in t.~(flow)}
fact {all p: chopP|one t: Pick|t in p.flow}
fact {all t: Pick|one p: chopP|p in t.~(flow)}
fact {all p: chopP|one t: Down|t in p.flow}
fact {all t: Down|one p: chopP|p in t.~(flow)}
.....
fact {all t: Pick|one p: chopP|p in t.flow}
fact {all p: chopP|one t:Pick|t in p.~(flow)}
fact {all t: Pick|one p:phP|p in t.flow }
fact {all p: phP|one t: Pick|t in p.~(flow)}
fact {all t: Pick|one p: eatP|p in t.flow }
fact {all p: eatP|one t: Pick|t in p.~(flow)}
.....

```

Finally, translate the fire conditions of all transitions as a predicate. The constraints for each transition include a pre-condition that is defined based the flow relation of the transition and the guard conditions of the transition, support functions, and a post-condition to define the effect of the firing.

```

-- pre-condition of transition Pick
pick in c.flow and pick in p.flow
and #(pick.~(flow)) = 2
and e in pick.flow and #pick.flow = 1
and #p.token.phi>0 and #c.token.right > 0
and #c.token.left > 0
and #(p.token.phi&c.token.right)>0
and some x:Int in c.token.right and some y:Int
in c.token.left and x = mod(y,5)
.....
-- post-condition of transition Pick
some x: Int in c.token.right and some y:Int in
c.token.left
and p.token.phi = p.token.phi - x
and c.token.left = c.token.left - y
and c.token.right = c.token.right - x
and e.token.phi = e.token.phi + x
and e.token.left = e.token.left + y
and e.token.right = e.token.right + x
-- pre and post-condition of transition Down
.....

```

Important properties such as safe or reachability can be defined as assertions to be analyzed by Alloy analyzer. Check the predicate *fire* with specified scope such as 6, Alloy analyzer will find instances for the model, which can be used for checking the original PrT net.

V. RELATED WORK

Software architecture has become an essential part in almost every phase of software development lifecycle [5]. Therefore, many researchers have proposed approaches and built tools for modeling and analyzing software architecture. In order to improve rigorousness of the analysis of software architecture and confidence of the quality of the architectural model, a

variety of formal modeling and analysis approaches and tools have been introduced during past two decades. Garlan [5] has summarized the representative results of formal modeling and analysis of software architecture. Allen and Garlan [2] described a formal basis for an architectural connection, which has become the one of the most important work on formal modeling of software architecture. Model checking has been reported for formally analyzing software architecture. In [7], He and *et. al.* reported approaches for formally analyzing Petri nets using model checking and formal proof techniques. Ding and He proposed an approach for modeling checking a type of high level Petri nets in [4]. Several other researchers defined an executable semantics for software architectural modeling and analysis through simulation and/or formal verification [13]. For example, formal specification language Rapide [12] supports simulation, Chemical Abstract Machine [9] and Wright [1] support limited formal verification. However, few work on testing of software architecture have been reported [15] due to its nature of informal and non-executable of architecture models in general. Zhu and He [19] proposed a methodology for testing design and architectural models in high level Petri nets. Model based testing uses results from testing architectural models for testing their corresponding implementations. For example, model level tests for testing an architecture are transformed into program level tests for testing its implementation [17][19]. The approach discussed in this paper is used for analyzing software architecture in Petri nets via naturally combing informal analysis techniques like software testing and formal analysis techniques like bounded model checking in two-phase analysis. The integration of bounded model checking with model based testing improves the rigorousness of model-based testing so that to improve the confidence of the correctness of important properties holding in the architecture. Although the pub-sub architecture has been widely implemented in many software systems, formal modeling and analyzing of its architecture is difficult. Garlan, Khersonsky and Kim [6] introduced a reusable generic framework for modeling and checking the model using model checker SMV. Kim and Garlan [11] also investigated how to analyze software architecture using Alloy. The approaches introduced in the two papers are similar to the approach of bounded model checking of architectural models discussed in this paper. The technique was also extended for model-based testing to improve the analysis performance and effectiveness.

VI. SUMMARY AND FUTURE WORK

In this paper, we presented an approach for modeling and analyzing software architecture through studying the pub-sub architecture. The approach is designed as a two-phase process to ensure it is both practical and rigorous for analyzing software architectures in PrT nets. In the first phase, a PrT net model is analyzed using model-based testing techniques including simulation, model checking and testing with tool MISTA. The bounded model checking is conducted by converting a PrT net into an Alloy model inputting to model analyzer Alloy. Then model-based test cases are generated from the checked model for selected test coverage criteria and finally they are converted into the program level tests for testing the corresponding implementation. Model-based software testing with MISTA has been introduced in other

publications [14][17], but the focus of this paper is on how to integrate bounded model checking into the process. The process of modeling and analysis of software architecture was illustrated by modeling and analyzing the pub-sub architecture. The approach is useful for analyzing software architecture in general, and also provides a framework for modeling and analyzing variety versions of pub-sub architecture. We plan to develop a tool to automate the transformation from a PrT net to an Alloy model.

ACKNOWLEDGMENTS

This research is supported in part by award #CNS-1262933 from the National Science Foundation. Junhua Ding's research was also partially supported by the guest professorship grant from school of computer sciences at China University of Geosciences.

REFERENCES

- [1] Acme, <http://www.cs.cmu.edu/~acme/>, last accessed on March 10, 2015.
- [2] R. Allen, D. Garlan. "A formal basis for architectural connection." ACM TOSEM 6 (3), pp. 213-249, 1997.
- [3] Alloy: <http://alloy.mit.edu>, last accessed on March 10, 2015.
- [4] J. Ding, X. He. "Formal Specification and Analysis of an Agent-Based Medical Image Processing System." Intl. Journal of SEKE, Vol. 20, No. 3, pp. 1 - 35, 2010.
- [5] D. Garlan, "Formal Modeling and Analysis of Software Architecture: Components, Connectors, and Events", in Formal Methods for Software Architectures, LNCS, Vol. 2804, pp. 1 -24, 2003.
- [6] D. Garlan, S. Khersonsky, and J.S. Kim, "Model Checking Publish-Subscribe Systems", Proc. of SPIN 03, Portland, Oregon, 2003.
- [7] X. He, H. Yu, T. Shi, J. Ding, and Y. Deng, "Formally Specifying and Analyzing Software Architectural Specifications Using SAM", Journal of Systems and Software, vol.71, no.1-2, pp.11-29, 2004, 1994.
- [8] P. Hens, M. Snoeck, G. Poels, D. B. Manu, "A petri net formalization of a publish-subscribe process system", FBE Research Report KBI_1114, K.U.Leuven - Faculty of Business and Economics, June 2011.
- [9] P. Inverardi, A. Wolf. "Formal specification and analysis of software architectures using the chemical abstract machine model." IEEE TSE, 21 (4), 373-386, 1995.
- [10] D. Jackson, "Software Abstractions: Logic, Language and Analysis", the MIT Press, 2012.
- [11] J. S. Kim, and D. Garlan, "Analyzing architectural styles", Journal of Systems and Software, 83(2010), pp. 1216-1235, 2010.
- [12] D. C. Luckham, J. Kenney, et al. "Specification and analysis of system architecture using rapide." IEEE TSE 21 (4), 336-355, 1995.
- [13] N. Medvidovic, R. Taylor, 2000. "A classification and comparison framework for software architecture description languages". IEEE TSE 26 (1), 70-93, 2000.
- [14] MISTA, <http://cs.boisestate.edu/~dxu/research/MBT.html>, last accessed on March 12, 2015.
- [15] D. Richardson, A. Wolf. "Software testing at the architectural level." In: Proc. of the 2nd Intl. Soft. Architecture Workshop. pp. 68-71, 1996.
- [16] J. A. Robles, G.A. Solano, "Modeling Petri nets using Alloy," TENCON 2012 - 2012 IEEE Region 10 Conference , vol., no., pp.1,6, 19-22 Nov. 2012.
- [17] D. Xu, "A Tool for Automated Test Code Generation from High-Level Petri Nets". 32nd Int. Conf. on Apps. and Theory of Petri Nets , Newcastle, UK, June 20-24, 2011.
- [18] D. Xu, D., K. E. Nygard, "Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets". IEEE TSE. 32(4), 265-278, 2006.
- [19] H. Zhu, and X. He, "A methodology of testing high-level petri nets". Journal of Information and Software Technology. v44, pp. 473-489, 2002.

Flexible and Extensible Runtime Verification for Java

Chengcheng Xiang¹, Zhengwei Qi¹, and Walter Binder²

¹Shanghai Jiao Tong University, Shanghai, China
Email: {xiangchengcheng, qizhwei}@sjtu.edu.cn

²Università della Svizzera italiana (USI), Switzerland
Email: {walter.binder}@usi.ch

Abstract—Runtime verification validates the correctness of a program’s execution trace. Much work has been done on improving the expressiveness and efficiency of runtime verification. However, current approaches require static deployment of the verification logic and are often restricted to a limited set of events that can be captured and analyzed, hindering the adoption of runtime verification in production systems. A popular system for runtime verification in Java, JavaMOP (Monitor-Oriented Programming in Java), suffers from the aforementioned limitations due to its dependence on AspectJ, which supports neither dynamic weaving nor an extensible join-point model. In this paper, we extend the JavaMOP framework with a dynamic deployment API and a new MOP specification translator, which targets the domain-specific aspect language DiSL instead of AspectJ; DiSL offers an open join-point model that allows for extensions. A case study on lambda expressions in Java8 demonstrates the extensibility of our approach. Moreover, in comparison with JavaMOP using load-time weaving, our implementation reduces runtime overhead by 21%, and heap memory usage by 16%, on average.

Keywords—Runtime verification; Monitor-Oriented Programming (MOP); dynamic program analysis; dynamic deployment

I. INTRODUCTION

Runtime verification [1], [2] is a method that dynamically checks specific properties of an executing system both in testing and production environments. Compared with traditional verification approaches, such as model checking [3] and automated theorem proving [4], runtime verification reduces the state space by concentrating on the actual execution trace and eliminates the fallibility of formally modeling a system. In recent years, a lot of research has aimed at making runtime verification a practical way to improve program reliability [1], [5], [6].

A lot of techniques and tools have been developed for runtime verification of Java programs. Early research, including Java-MaC [7] and Hawk/Eagle [5], is focused on developing expressive logics for property description. Recently, JavaMOP [6] effectively reduces the runtime overhead thanks to an efficient management of monitors. Moreover, static program analysis techniques are used to reduce the amount of inserted instrumentation code [8].

However, a lack of flexibility and extensibility has prevented these techniques from becoming widely used in practice. Flexibility is important for two reasons. On the one hand, runtime verification systems may introduce a significant overhead of more than 100% when monitoring multiple properties

simultaneously [9]. In some cases, such overhead may be inevitable, because checking multiple properties simultaneously may need to monitor a large number of events. Hence, it is necessary to verify properties sequentially, implying that code for event capture needs to be deployed and undeployed dynamically. On the other hand, as dynamic code evolution has been a timesaving way for development, property checkers should also be able to get updated dynamically. Moreover, since most runtime verification tools for Java, such as Tracematches [10] and JavaMOP [6], use AspectJ [11] as their instrumentation back-end, the categories of events are restricted to the AspectJ pointcuts, which can only be extended by modifying the AspectJ compiler. As shown e.g. in [12], the AspectJ join-point model is not well suited for dynamic program analysis.

In this paper, we present a flexible and extensible runtime verification framework for Java, MOP-DiSL. Our approach is based on JavaMOP [6], a framework for Monitoring-Oriented Programming for Java, and on DiSL [12], a domain-specific language for dynamic program analysis based on bytecode instrumentation. Our framework translates the MOP specification into DiSL code, and a new deployment API allows for flexible (un)deployment of instrumentation code at runtime. Extensibility of event categories is achieved through DiSL’s open join-point model, which we demonstrate with a case study on Java8 lambda expressions.

This paper makes the following contributions:

- We present MOP-DiSL, a novel runtime verification framework for Java that offers flexible dynamic (un)deployment and extensibility in terms of event types that can be captured.
- We conduct a case study on lambda expression-related properties in Java8 programs, and we add several new pointcuts to show the extensibility of MOP-DiSL.
- We evaluate MOP-DiSL by verifying four properties with the DaCapo benchmarks [13], showing that MOP-DiSL introduces significantly less runtime overhead and consumes less heap memory than JavaMOP with load-time weaving.

This paper is structured as follows. Section II provides background information on JavaMOP, AspectJ, and DiSL. Section III gives a motivating example. Section IV presents the design and some implementation details of our framework. Section VI evaluates our framework in comparison with JavaMOP. Finally, Section VII concludes.

```

List<Integer> l1 = new ArrayList<>();
List<Integer> l2 = new ArrayList<>();
...
Iterator<Integer> itr = l1.iterator();
while (itr.hasNext()) {
    l2.add(itr.next()*2);
}

```

Listing 1: Iterator example

```

l1.parallelStream()
    .map(e -> e*2)
    .forEach(e -> l2.add(e));

```

Listing 2: Stream example

II. BACKGROUND

A. JavaMOP

JavaMOP is an implementation of Monitor-Oriented Programming (MOP) that enables runtime verification on the Java platform. The implementation consists of two parts: a translator and a set of runtime libraries. The translator parses specifications of properties in the form of finite state machines (FSM), context-free grammars (CFG), extended regular expressions (ERE), and other logical formalism, and generates AspectJ code to monitor events. Using AspectJ to weave code restricts the flexibility of JavaMOP and the categories of events it can get. However, adding new event types to MOP specifications requires both extensions to the JavaMOP translator and to the AspectJ compiler.

Many optimization techniques have been proposed to improve the efficiency of the runtime libraries of JavaMOP. [6] adopts centralized and decentralized indexing algorithms to optimize the lookup process of monitors. In order to efficiently reclaim monitor instances that are bound to parameter objects, [14] proposes a lazy garbage-collection (GC) strategy. Other than the aforementioned generic techniques, optimizations are also performed for specific property patterns [15]. According to [16], the verification code only causes an average runtime overhead of 15% on the DaCapo benchmark. However, the overhead for simultaneously monitoring multiple properties remains prohibitive in practice [9], indicating that there is need for a more flexible approach to runtime verification.

B. AspectJ

Used by JavaMOP for event definition and instrumentation, AspectJ [11] is an aspect-oriented programming (AOP) extension to the Java programming language. AspectJ adopts pointcuts to select join points, which are execution points in a program flow, and advice to define the actions to be executed before, after, or around each join point. Common AspectJ implementations weave advice in two ways: pre-load weaving and load-time weaving. Dynamic AOP, supported by tools such as AspectWerkz [17], Prose [18], and HotWave [19], [20], also enables runtime weaving and runtime adaption of the aspect code. Such techniques may endow runtime verification with more flexibility; however, there are still restrictions on the extensibility of event categories.

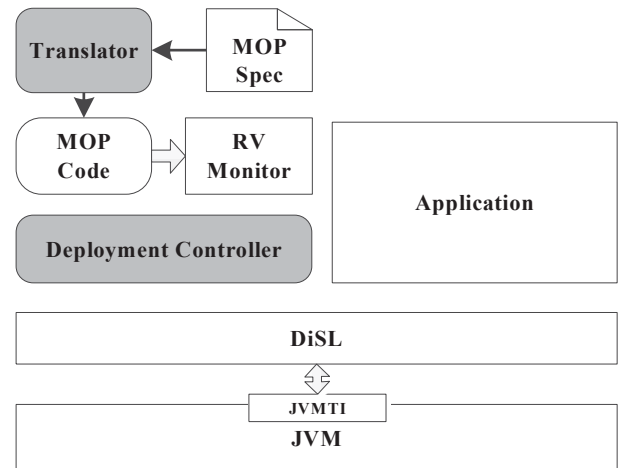


Figure 1: Framework Overview.

C. DiSL

As a tool for Java bytecode instrumentation, DiSL [12] is distinguished with several features. First, unlike AspectJ, DiSL supports an open join-point model, which means that any code region can be marked as a join point to trigger an event. Second, DiSL offers comprehensive bytecode coverage when weaving, including the Java core class library and dynamically generated code. Moreover, DiSL has been enhanced with the ability of dynamically deploying and undeploying analysis code [21]. Although DiSL possesses the necessary flexibility and extensibility for runtime verification, it lacks expressiveness to describe properties and efficient management of monitor instances.

III. MOTIVATING EXAMPLE

In this section, we present an example to show the necessity for flexibility and extensibility in runtime verification frameworks.

Listing 1 and Listing 2 demonstrate two code segments with the same function: doubling the integers in l1 and appending the outcome to l2. The difference resides in the way they realize it. In Listing 1, an iterator is used to traverse l1, while the code in Listing 2 utilizes a parallel stream (introduced in Java8) to eventually accelerate the process.

When runtime verification is applied to these two code segments, different properties are of concern. For Listing 1, as an iterator is created, we may be concerned about the hasNext property, which means hasNext() should always be called before the execution of next() on an iterator. For Listing 2, it is pointless to check the hasNext property, because the parallel stream internally traverses the list. Instead, since the statements in the lambda expressions are executed concurrently, it becomes necessary to verify whether they are thread-safe. In this example, the method call l2.add() is not safe.

Given a dynamic software update substituting Listing 1 with Listing 2 in a running system that should not be restarted, the properties for runtime verification should also be replaced dynamically. The lambda expressions pose an additional challenge, because there is no pointcut in current AspectJ to

```

event updatesource after(Collection c) :
    (call(* Collection+.remove*(..))
    || call(* Collection+.add*(..)) ) &&
    target(c) {}

```

Listing 3: Event example

```

@After(marker = CollectionRemoveMarker.class)
public void Updatesource0 (DynamicContext dc,
    ArgumentProcessorContext apc) {
    Collection c =
        (Collection)apc.getReceiver();
    RuntimeMonitor.updatesourceEvent(c);
}

@After(marker = CollectionAddMarker.class)
public void Updatesource1 (DynamicContext dc,
    ArgumentProcessorContext apc) {
    Collection c =
        (Collection)apc.getReceiver();
    RuntimeMonitor.updatesourceEvent(c);
}

```

Listing 4: DiSL example

capture the event that a lambda expression is called in a stream stage. More details about lambda expressions are discussed in Section V. These challenges demonstrate the need for flexibility and extensibility of runtime verification systems.

IV. FRAMEWORK DESIGN AND IMPLEMENTATION

In this section, we first give an overview of the framework architecture, and then present the design and implementation details of each part.

A. Overview

Figure 1 depicts an overview of MOP-DiSL. In the beginning, the translator generates MOP code in Java, with inputs of MOP specifications. Then the deployment controller deploys the MOP code dynamically (*i.e.*, instruments an application with the MOP code). The MOP code will capture sensitive events and generate corresponding monitors, which are managed by RV-Monitor, a runtime library from JavaMOP. The whole framework is based on DiSL, which relies on the JVMTI, a standard tool interface for the JVM. Hence, a high degree of portability of MOP-DiSL is achieved.

Flexibility and extensibility are achieved through the deployment controller and the translator separately. On the one hand, the deployment controller instruments, updates, and removes bytecode for event capture without pausing or restarting the target application. On the other hand, the translator generates DiSL code as instrumentation back-end, gaining extensibility from DiSL’s open join-point model.

B. Translator

The translator extends the original JavaMOP translator to generate DiSL code for instrumentation (instead of AspectJ). The JavaMOP translator parses event descriptions with the definition syntax of AspectJ advice, as shown in Listing 3, and the advice definitions are copied to the final aspect for event capture. Our translator takes the same input, but generates

Table I: Deployment Controller API

Interface	Description
deploy(property, scope)	Deploy MOP code of a property in a specific scope
undeploy(property, scope)	Undeploy MOP code of a property from a specific scope
redeploy(property, scope)	Update MOP code of a property in a specific scope

DiSL code for instrumentation. Listing 4 demonstrates DiSL code generated from the event definition in Listing 3, with an annotation to define the event, and a method to express the reactions to the event.

The main challenge for translating advice to DiSL lies in the composition of pointcuts. AspectJ offers a set of primitive pointcuts, which can be combined by “&&” and “||” to describe complex events. DiSL also offer a group of basic markers to mark code regions as join points. However, there is no easy way to combine these markers, except extending the marker class. Our translator combines basic markers by generating new markers, *e.g.*, CollectionRemoveMarker and CollectionAddMarker, which inherit from a basic MOP marker. To shorten the generated code, the base MOP marker is modeled as a pipeline of filters, each for one basic pointcut. An extended marker overrides several filters to express “&&” of these filters, while “||” will be translated into different markers. In order to generate markers for “||”, the translator first transforms the pointcut composition into a disjunctive normal form, and each clause will be converted to a marker.

New pointcuts can be easily added to define events by extending the translator and the base marker class. Since the marker class manipulates bytecode through ASM¹, a lightweight framework, it is very convenient to expand it with more filters, corresponding to more primitive pointcuts. We have added two pointcuts, *i.e.*, lambdaDef and thisLambda, to capture events about lambda expressions. More details are presented in Section V.

C. Deployment Controller

The deployment controller provides several interfaces to attain flexibility, which are listed in Table I. The three interfaces are declared with identical parameters, two string variables: property refers to a property name, such as *hasNext*, and scope refers to its deploying scope. The scope can be either a package name or a class name with wildcards support.

These interfaces are implemented based on the work in [21], which implemented dynamic bytecode instrumentation through the retransformClasses interface in JVMTI. In our current implementation, the controller interfaces are exposed through a network client, with which programmers can deploy or undeploy property monitors to a running system. However, these interfaces can also be called by optimized monitor programs which only enable property monitors when necessary.

V. CASE STUDY

In this section, we present a case study on checking whether unsynchronized collections are modified in a parallel

¹<http://asm.ow2.org/>

```

UnsafeForEach(Object capVar0, Consumer c){
    event create after(Object capVar0)
        returning(Consumer c) :
            lambdaDef() && args(capVar0){}
    event forEach before(Consumer c):
        call(* ParallelStream.forEach(..)
            && args(c) {}
    event update after(Consumer c, Object
        capVar0):
        (call(*
            (!SynchronizedCollection+).remove*(..))
        || call(*
            (!SynchronizedCollection+).add*(..))
        && target(capVar0) && thisLambda(c){}
    ere : create forEach update
    @match {
        ...
    }
}

```

Listing 5: UnsafeForEach

stream, of which a violation instance is shown in Listing 2 as the motivation example.

Listing 5 demonstrates the property definition in the form of MOP specification. The property is named as UnsafeForEach with two parameters defined in the event definitions, *i.e.*, `capVar0` and `c`. Three types of events are captured by the verification process, and ERE is adopted to express the targeting event trace. When a matching trace is detected, an error message is printed.

The first event is about creating a lambda expression, defined with a new pointcut, *i.e.* `lambdaDef`. In Oracle’s Java8 implementation, lambda expressions are translated using the bytecode instruction `invokedynamic`. This instruction causes the lambda metafactory to be invoked and generates a lambda object implementing the `Consumer` functional interface. The `lambdaDef` pointcut selects all `invokedynamic` instructions, checks whether they call the lambda metafactory, exposes captured variables and the return value through the corresponding pointcuts `args` and `returning`. In this case, there is only one captured variable, *i.e.*, the collection object. However, more objects can be exposed by `args()`, if more variables are captured by the lambda expression.

The second event is calling the `forEach` method of a parallel stream instance. The `forEach` method takes a `Consumer` object as its parameter, and registers the object to the stream. The new `ParallelStream` pointcut is added for that a parallel stream is also an instance of `Stream`, which means it is impossible to pick out a parallel stream only with current type check.

The last event happens when an unsynchronized collection is updated in a lambda expression. The new pointcut `thisLambda` ensures that the updating executes in a lambda context, and exposes the lambda object `c`. It is worth noting that the lambda object cannot be exposed by AspectJ with the pointcut execution `(“*lambda*”) && this(c)`, since the lambda body is implemented as a static method and `this()` will return null. Another way is by the pointcut `call(“*lambda*”) && target(c)`, which also selects nothing because the lambda expression is called by the parallel stream, a class in the Java core library that cannot be woven by AspectJ. MOP-DiSL takes advantage

```

public class LambdaDefMarker extends
    MOPBaseMarker {
    public List<MarkedRegion> mainFilterPipe(
        AbstractInsnNode insn){
        List<MarkedRegion>
            regions=super.mainFilterPipe(insn);
        filterLambda(regions, insn);
        return regions;
    }
    void filterLambda(regions,
        AbstractInsnNode insn){
        if (insn instanceof
            InvokeDynamicInsnNode){
            if (matchIndy(indy))
                regions.add(new
                    MarkedRegion(insn, insn));
        }
    }
    //This will be overridden by the generated
    code
    boolean matchIndy(AbstractInsnNode insn){
        return false;
    }
}

```

Listing 6: LambdaDefMarker

of DiSL’s ability to instrument the Java core library to capture the call event and expose the target object.

Adding these pointcuts consists of two steps: implementing new markers and extending the translator. Listing 6 shows the marker code for implementing `lambdaDef` pointcut. The new marker is introduced as a subclass of the base marker, adding a new filter process, *i.e.* `filterLambda`, to the main filter pipeline. What `filterLambda` does is simply checking the type of current instruction and calling `matchIndy()` to decide whether the instruction should be marked for weaving. The translator needs to be extended to generate a subclass of `LambdaDefMarker` with a proper overriding of the method `matchIndy` according to the pointcut definition, which is much simpler than modifying the AspectJ weaver.

VI. EVALUATION

In this section, we evaluate the runtime overhead and heap memory consumption of MOP-DiSL, and compare it with JavaMOP.

A. Experimental Settings

We set up the experimental environment on a Dell Optiplex 980 machine with 8GB memory and an Intel 3.20GHz i5 CPU. We use JDK 1.8 and AspectJ 1.8 for compilation on a 64 bits Debian 7 system. For AspectJ, load-time weaving is adopted, since DiSL uses dynamic weaving. DaCapo 9.12 [13], a popular benchmark suite for Java, is utilized to evaluate the runtime overhead and heap memory usage. The benchmark is run for 10 iterations and the result of the first iteration is also counted, as the code weaving only happens in the first iteration when classes are first loaded. Four well-known properties are verified in the evaluation:

- 1) `HasNext`: method `hasNext()` should be called before calling `next()` for an iterator;

Table II: Execution time and overhead percentage for JavaMOP and MOP-DiSL with different properties

Benchmark	origin sec.	HasNext				UnsafelFter				UnsafeMapItr				UnsafeFileWriter			
		JavaMOP-LW sec.	ovh.(%)	MOP-DiSL sec.	ovh.(%)	JavaMOP-LW sec.	ovh.(%)	MOP-DiSL sec.	ovh.(%)	JavaMOP-LW sec.	ovh.(%)	MOP-DiSL sec.	ovh.(%)	JavaMOP-LW sec.	ovh.(%)	MOP-DiSL sec.	ovh.(%)
avrora	4.26	6.14	44.27	5.74	34.92	21.96	416.02	17.60	313.54	8.27	94.43	8.51	99.88	5.88	38.18	5.70	33.98
batik	2.25	2.75	22.19	2.59	15.08	2.96	31.64	3.35	48.82	2.87	27.46	2.92	29.78	3.90	73.36	3.50	55.80
fop	1.17	2.08	76.95	2.02	71.97	2.34	99.44	2.06	75.70	2.50	112.97	3.09	163.44	2.32	97.75	1.83	55.67
h2	6.94	7.84	13.01	7.45	7.38	7.86	13.33	7.10	2.38	8.53	22.91	10.13	46.00	8.80	26.89	8.62	24.29
lucene	5.77	6.74	16.88	6.24	8.22	7.41	28.41	6.39	10.71	7.92	37.24	7.23	25.30	9.58	65.95	8.61	49.26
lucene	1.27	1.41	11.36	1.41	11.09	1.45	14.07	1.40	10.28	1.50	18.15	1.41	11.03	1.94	52.87	1.80	41.84
lucene	2.35	2.62	11.27	2.65	12.49	3.04	29.16	2.82	19.79	2.60	10.39	2.79	18.49	3.41	44.94	3.14	33.52
lucene	4.11	5.95	44.83	5.66	37.81	8.64	110.28	7.79	89.67	10.27	149.89	8.25	100.87	6.12	48.94	5.48	33.27
lucene	6.17	6.38	3.43	6.27	1.71	6.25	1.39	6.04	1.02	6.27	1.65	6.27	1.68	8.35	35.47	7.65	24.05
lucene	3.83	5.27	37.63	4.17	8.93	5.48	43.20	4.89	27.65	5.66	47.77	4.79	25.02	6.41	67.52	4.81	25.77
lucene	6.84	18.53	171.18	17.77	160.06	14.83	117.02	13.55	98.28	21.13	209.12	20.22	195.83	9.51	39.10	9.14	33.67
lucene	12.31	18.19	47.81	17.99	46.17	24.25	97.00	23.59	91.62	25.32	105.69	26.35	114.07	14.88	20.87	14.98	21.71
lucene	2.91	3.12	7.15	3.06	5.17	3.59	23.47	3.55	21.97	3.18	9.42	3.40	16.81	3.55	22.25	3.94	35.56
geo.mean			23.63		16.55		38.11		26.38		34.95		36.21		44.20		34.41

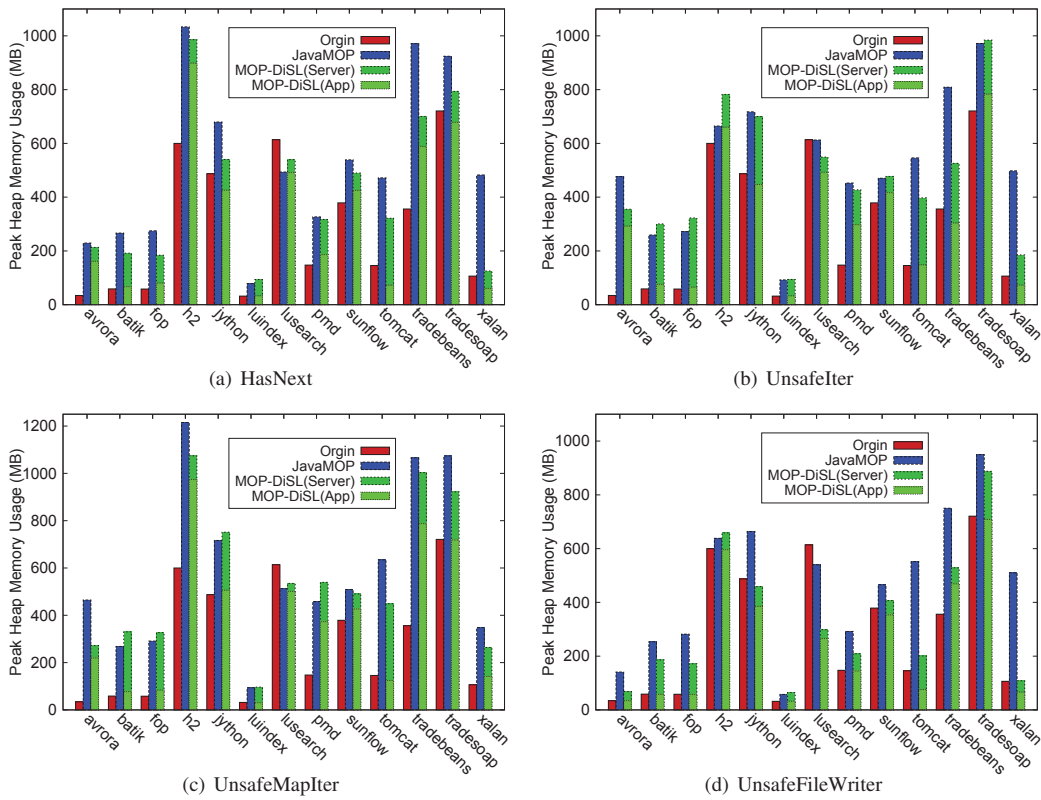


Figure 2: Peak heap memory usage. MOP-DiSL (Server) refers to the memory consumption in the DiSL instrumentation server, which runs in another Java process. The actual runtime memory for application + analysis is noted as MOP-DiSL (App).

- 2) UnsafelFter: a Collection should not be updated when its iterator is accessed;
- 3) UnsafeMapItr: a Map should not be updated when its iterator is accessed;
- 4) UnsafeFileWriters: no writing is allowed after a file is closed.

is a bug when it is run on Java8². We do not evaluate the performance of the lambda property for there is currently no standard benchmark with much use of the Java8 stream API.

B. Runtime Overhead

Table II displays the execution time and percent overhead of JavaMOP-LW and MOP-DiSL. On average (geometric mean), MOP-DiSL incurs less overhead than JavaMOP-

The results are presented with JavaMOP-LW referring to JavaMOP using AspectJ load-time weaver. The results of eclipse benchmark in DaCapo are not presented because there

²<http://mail.openjdk.java.net/pipermail/aarch64-port-dev/2014-February/000844.html>

LW for property HasNext, UnsafeIter, and UnsafeFileWriter. MOP-DiSL achieves more overhead reduction for applications with higher overhead, which have more instrumented code executed, and the reason is that code instrumented by DiSL is more efficient than AspectJ. However, weaving code in an isolated instrumentation server, DiSL spends more time on communicating with the server so that more time on weaving. For UnsafeMapIter, there are more event types, which means more code needs to be instrumented, resulting that more communication with the server is needed for DiSL. Consequently, MOP-DiSL causes similar overhead with JavaMOP-LW and more overhead for some benchmarks, *i.e.*, avrora, batik, fop, h2, lusearch, tradesoap, and xalan. Overall, MOP-DiSL causes 21% less runtime overhead than JavaMOP-LW.

C. Memory Consumption

Figure 2 illustrates the peak Java heap usage when the four properties are verified. The memory consumption of MOP-DiSL consists of the server part and the application part. The instrumentation server, which does code weaving, is run in a separated process and in theory can be run in another physical machine. We accumulate the memory usage mainly to show that even combining both application and instrumentation server, our approach still outperforms JavaMOP by 16% in terms of heap consumption (evaluated by geometric mean). The application part of MOP-DiSL consumes 54% lower Java heap than JavaMOP on average. This feature benefits the application process with less GC time. For some benchmarks, such as lusearch, JavaMOP and MOP-DiSL even cause lower peak heap consumption than the original application. This is mainly due to different GC behavior—with more memory usage in the runtime analysis, GC may be triggered more often and hence keep the memory consumption at a lower level.

VII. CONCLUSION

In this paper, we present a novel framework, MOP-DiSL, to achieve flexibility and extensibility in runtime verification for Java. A deployment API is designed for flexibility and a new MOP translator is devised with extensibility. We demonstrate a case study on adding event types to check properties associated with lambda expressions in Java8, which requires extensibility. Evaluation results show that our framework causes less runtime overhead and lower peak heap usage than JavaMOP with AspectJ load-time weaving. As a result, our framework achieves flexibility and extensibility with no more runtime and memory overhead, making runtime verification more practical in real-world settings.

ACKNOWLEDGMENT

This work is supported by NSFC (No. 61272101), National R&D Infrastructure and Facility Development Program (No. 2013FY111900), NRF Singapore CREATE Program E2S2, Sino-Swiss Science and Technology Cooperation (SSSTC), and Shanghai Key Laboratory of Scalable Computing and Systems.

REFERENCES

[1] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.

[2] C. Kloukinas, G. Spanoudakis, and K. Mahbub, "Estimating event lifetimes for distributed runtime verification," in *SEKE*, pp. 117–122, 2008.

[3] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[4] M. Fitting, *First-order logic and automated theorem proving*. Springer Science & Business Media, 1996.

[5] M. d’Amorim and K. Havelund, "Event-based runtime verification of Java programs," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, pp. 1–7, ACM, 2005.

[6] F. Chen and G. Roşu, "Mop: an efficient and generic runtime verification framework," in *ACM SIGPLAN Notices*, vol. 42, pp. 569–588, ACM, 2007.

[7] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan, "JavaMaC: a run-time assurance tool for Java programs," *Electronic Notes in Theoretical Computer Science*, vol. 55, no. 2, pp. 218–235, 2001.

[8] E. Bodden, P. Lam, and L. Hendren, "Clara: A framework for partially evaluating finite-state runtime monitors ahead of time," in *Runtime Verification*, pp. 183–197, Springer, 2010.

[9] Q. Luo, Y. Zhang, C. Lee, D. Jin, P. O. Meredith, T. F. Serbanuta, and G. Rosu, "RV-Monitor: Efficient parametric runtime verification with simultaneous properties," in *Proceedings of the 14th International Conference on Runtime Verification (RV’14)*, LNCS, September 2014.

[10] C. Allan, P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, O. Lhoták, O. De Moor, D. Sereni, G. Sittampalam, and J. Tibble, "Adding trace matching with free variables to AspectJ," in *ACM SIGPLAN Notices*, vol. 40, pp. 345–364, ACM, 2005.

[11] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of AspectJ," in *ECOOP 2001*, pp. 327–354, Springer, 2001.

[12] L. Marek, A. Villazón, Y. Zheng, D. Ansaloni, W. Binder, and Z. Qi, "DiSL: a domain-specific language for bytecode instrumentation," in *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*, pp. 239–250, ACM, 2012.

[13] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, *et al.*, "The DaCapo benchmarks: Java benchmarking development and analysis," in *ACM Sigplan Notices*, vol. 41, pp. 169–190, ACM, 2006.

[14] D. Jin, P. O. Meredith, D. Griffith, and G. Rosu, "Garbage collection for monitoring parametric properties," in *ACM SIGPLAN Notices*, vol. 46, pp. 415–424, ACM, 2011.

[15] P. Meredith and G. Rosu, "Efficient parametric runtime verification with deterministic string rewriting," in *Proceedings of 28th IEEE/ACM International Conference. Automated Software Engineering (ASE’13)*, p. NA, IEEE/ACM, May 2013.

[16] D. Jin, P. O. Meredith, C. Lee, and G. Roşu, "JavaMOP: Efficient parametric runtime monitoring framework," in *Proceeding of the 34th International Conference on Software Engineering (ICSE’12)*, pp. 1427–1430, IEEE, 2012.

[17] J. Bonér, "Aspectwerkz: Dynamic AOP for Java," in *Invited talk at 3rd International Conference on Aspect-Oriented Software Development (AOSD)*, Citeseer, 2004.

[18] A. Nicoară and G. Alonso, "Dynamic AOP with Prose," in *1st International Workshop on Adaptive and Self-Managing Enterprise Applications*, pp. 125–138, 2005.

[19] A. Villazón, W. Binder, D. Ansaloni, and P. Moret, "Advanced runtime adaptation for Java," in *Proceedings of the Eighth International Conference on Generative Programming and Component Engineering, GPCE ’09*, pp. 85–94, ACM, Oct. 2009.

[20] A. Villazón, W. Binder, D. Ansaloni, and P. Moret, "HotWave: creating adaptive tools with dynamic aspect-oriented programming in Java," in *ACM Sigplan Notices*, vol. 45, pp. 95–98, ACM, 2009.

[21] Y. Zheng, L. Bulej, C. Zhang, S. Kell, D. Ansaloni, and W. Binder, "Dynamic optimization of bytecode instrumentation," in *Proceedings of the 7th ACM workshop on Virtual machines and intermediate languages*, pp. 21–30, ACM, 2013.

Improving the Accuracy of Integer Signedness Error Detection Using Data Flow Analysis

Hao Sun, Chao Su, Yue Wang, Qingkai Zeng

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China

Email: {shqing, suchao1991}@gmail.com, wxywang89@163.com, zqk@nju.edu.cn

Abstract—Integer signedness error can be exploited by attackers to cause severe damages to computer systems. Despite of the significant advances in automating the detection of integer signedness errors, accurately differentiating exploitable and harmful signedness errors from unarmful ones still remains an open problem. In this paper, we present the design and implementation of *SignFlow*, an instrumentation-based integer signedness error detector to reduce the reports for unarmful signedness errors without sacrificing the completeness (i.e. no false negatives). *SignFlow* utilizes static data flow analysis to identify *unarmful integer signedness conversions* from the view of where the operands originate and whether the data after conversions can propagate to security-related operations, and then inserts security checks for the remaining conversions so as to accomplish runtime protection. We evaluated *SignFlow* on 7 real-world harmful integer signedness bugs, SPECint 2006 benchmarks together with 5 real-world applications. Experimental results show that *SignFlow* successfully detected all harmful integer signedness bugs and achieved a reduction of 41% in false positives over *IntFlow*, the state-of-the-art signedness error detector.

Keywords—integer signedness error, data flow analysis, instrumentation, sanitization

I. INTRODUCTION

The C/C++ programming language implements the *signedness* of integer types, including signed and unsigned. An *integer signedness error* occurs when a signed integer is interpreted as unsigned, or vice-versa. In two-complement representation, such conversions cause the sign bit to be interpreted as the most significant bit (MSB) or conversely, hence -1 and $2^{32} - 1$ are misinterpreted to each other on 32-bit machines. Because such misinterpretation cannot overwrite memory directly, adversaries usually leverage *security-related operations* (e.g., bound checks, memory allocations and loops) to exploit integer signedness errors indirectly. For instance, Listing 1 shows a typical signedness error in Mumble [1]. Lines 4 to 5 are the patch for this bug. In the original buggy code, variable `decodedSamples` is used to denote the amount of decoded samples, and it would be assigned with small negative values when `opus_decode_float()` encounters an error. Note that such small negative integers indicate the error condition. Then `decodedSamples` is converted to unsigned integer, i.e. `inlen`, which becomes close to `UINT_MAX`. Later `inlen` is used as the buffer length in `speex_resample_process_float()` and buffer overflow occurs due to such inadvertently large buffer length.

Listing 1. Patched code for CVE-2014-0045 in Mumble

```
1 int decodedSamples = opus_decode_float(opusState
, NULL, ...);
2 ...
3 spx_uint32_t inlen = decodedSamples;

4+ if(inlen > 0x7fffffff)
5+     return error;

6 if (srs && bLastAlive)
7     speex_resampler_process_float(srs, 0,
fResamplerBuffer, &inlen, ...);
```

During the past years, researchers have developed various techniques to address this problem. A classic approach is to insert *security checks* around integer signedness conversions to catch signedness errors at runtime. Instrumented programs can react to a signedness error by logging the event or terminating the execution. Many existing techniques, such as RICH [2], IOC [3], AIR [4] and RA [5], consider *all* integer signedness conversions in a subject program as potential signedness error sites and instrument all of them. The *instrument-all* techniques guarantee to detect all the runtime signedness errors; however, this safety has a price: they might report unarmful signedness errors as critical ones in real-world scenarios, i.e. producing false positives.

IntFlow [6] aims to eliminate the instrumentation for some integer signedness conversions if they originate from trusted source. The intuition is that such trusted integer signedness conversion cannot be controlled by attackers even if signedness error occurs. Hence, *IntFlow* could reduce a number of false positives produced by instrument-all techniques. However, *IntFlow* only considers where a signedness error originates, but doesn't track how it would be used afterward. *IntFlow* would produce false positives inevitably in the following cases: 1) the integer data after conversion is unused afterward, or propagates to uncritical program locations; 2) experienced developers often anticipate the possibility of signedness errors such that they add *sanitization routines* after these sites; 3) programmers use signedness errors intentionally for performance optimization or code compactness in many situations.

To further improve the accuracy of integer signedness error detection, we develop a novel runtime signedness error detector, named *SignFlow*, which features the capacities of detecting all harmful signedness errors (*complete*) and by-passing as many unarmful ones as possible (*precise*) with

(DOI reference number: 10.18293/SEKE2015-123)

acceptable performance loss (*practical*). In SignFlow, we not only consider where integer signedness conversions originate (as IntFlow does), but further track how they would be used afterward. The main intuition behind our approach is that integer signedness errors become unarmful if they are unused afterward, or used at uncritical program locations such as *printf()*, or get sanitized before flowing into security-related operations. As such, SignFlow could avoid the reports for more unarmful signedness errors, compared to IntFlow.

Our contributions are highlighted as follows.

- We define unarmful integer signedness errors from the perspective of where they originate and how they are used afterward;
- We propose and implement a novel instrumentation-based integer signedness error detector, SignFlow, as an extension of the GCC compiler [7], aiming to improve the accuracy of integer signedness error detection. SignFlow first exploits static data flow analysis to identify unarmful integer signedness conversions and then inserts security checks for the remaining potentially harmful sites;
- To demonstrate the effectiveness, we applied SignFlow to 7 harmful signedness bugs, SPECint 2006 benchmarks and 5 real-world applications. Experimental results show that SignFlow detected all 7 harmful errors and bypassed about 41% (46 out of 111) unarmful ones that the state-of-the-art integer signedness error detector, i.e. IntFlow, produced. Besides, as about 7.73% more integer signedness conversions were identified as unarmful at static analysis phase, SignFlow reduced the runtime overhead by 49.62% over IntFlow.

II. APPROACH

One key consensus in existing techniques is that an integer signedness error becomes harmful (or critical) if it satisfies both the following two conditions: *T1: the right-hand operand in signedness conversion originates from un-trusted source, i.e. controlled by users; T2: the misinterpreted data may propagate to security-related operations, i.e. sinks.*

T1 allows attackers to control this signedness error and determine the misinterpretation according to their own intentions, and T2 provides attackers an interface to exploit this signedness error so as to conduct malicious operations. Conversely speaking, we argue that *an integer signedness error can be treated as unarmful if it violates either T1 or T2.* In the following, we first present data flow patterns of unarmful signedness errors from the perspective of violating T1 or T2, and then introduce our approach briefly.

A. Unarmful Integer Signedness Errors

Definition 1: An integer signedness error is unarmful if it satisfies one of the following three conditions:

1. *the right-hand operand in signedness conversion is constant or originates from constant values;*
2. *the integer data after conversion is unused afterward, or propagates to uncritical program locations;*

TABLE I. POST-CONDITION TEST FOR SIGNEDNESS CONVERSIONS

Signedness Conversion	Post-condition Test
int x; unsigned int y; x = (int) y;	x < 0
unsigned int x; int y; x = (unsigned int) y;	x > 0x7fffffff

TABLE II. SANITIZATION OPERATOR FOR SIGNEDNESS ERRORS

Op Type	Basic Form	Influence
bitwise-and	res = a & b, and b < 0x80000000	Erased
modulo	res = a % b, and b < 0x80000000	Erased
left-shift	res = a << b	Replaced
right-shift	res = a >> b	Replaced
bitwise-xor	res = a ⊕ b	Erased
bitwise-not	res = ~ a	Erased

3. *the integer data after conversion gets sanitized before it propagates to security-related operations.*

1) *Trusted Source:* The integer signedness error under condition 1.1 means that, the concrete value of source operand can be determined at compile-time. Hence, attackers cannot control this signedness error via providing crafted inputs. In other words, attackers have no chance to exploit such signedness errors. We call this data flow pattern of unarmful signedness errors as P_{cst} for short.

As shown below we adopt the code snippet from [6] to present an example, in which developers intentionally rely on signedness error mainly for performance reasons. It casts -1 into unsigned type to obtain the largest number that unsigned type can represent. Since the source operand is constant, satisfying P_{cst} , this signedness error is unarmful.

```
UINT_MAX = (unsigned int) -1;
```

2) *Uncritical Program Location:* The integer signedness error under condition 1.2 denotes that, the integer data after conversion is relatively not as critical as other integers. This shuts off the interface for attackers to jeopardize the whole program even if attackers can control this misinterpreted data. We call this data flow pattern of unarmful signedness errors as P_{uncrit} for short. Note that library function calls such as *printf()* and *fprintf()*, are typical uncritical locations.

3) *Sanitization:* According to our Definition 1, the paths of misinterpreted data to sinks can be cut off or stopped if there exist so-called sanitizations. Here, we give the following two kinds of sanitizations.

Experienced developers often anticipate the possibility of integer signedness errors such that they add *sanitization routines* after potential signedness error sites to prevent misinterpreted data from affecting further program execution. As shown in Table I, post-condition test [3][8] is the widely used sanitization routines for signedness errors. The patch for CVE-2014-0045, i.e. lines 4 to 5 in Listing 1, gives an example. Once signedness error occurs at line 3, the security check at line 4 would catch this misinterpretation, cutting off the flow to sink (i.e. line 7). As post-condition test is of fixed patterns, they can be identified statically. Furthermore, post-condition test is sound enough to catch signedness errors. Hence, we adopt post-condition test as one kind of sanitizations for signedness errors, and we call such data flow pattern as P_{post} for short.

Besides, many operators can remove or clean up the sign bit for signed type or the MSB for unsigned type. In these

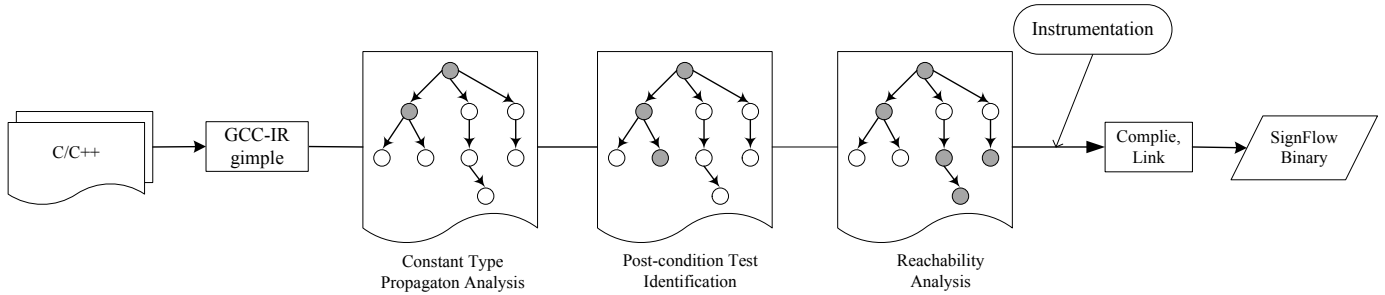


Fig. 1. Overall architecture of SignFlow

cases, the misinterpreted data gets sanitized since the bit, that signedness error matters, is *replaced by another benign bit or erased* after such operators. We call these operators *SantzOp* for short. Table II shows a serial of *SantzOp*, including the type, the basic form and the influence on the sign bit/MSB. Note that a denotes the misinterpreted integer. Here, we argue that *after going through SantzOp, misinterpreted data is less likely to be used in exploitation attempts by attackers, as the crucial bit, i.e. the sign bit/MSB, is replaced with another benign bit or erased by SantzOp*. We call such data flow pattern as P_{op} for short.

Take the following code snippet as an example. It is from *pp.c*, 400.perlbench in SPECint 2006. Implicit signedness conversion occurs at line 2332; however, the bitwise-and operators at line 2335 can erase the sign bit. Hence, such signedness error is unharmsful. In fact the *SantzOp* at line 2335 is used here to achieve the effect of selecting specific bits of `flags`.

```

struct STRUCT_SV{
    void * sv_any;
    U32 sv_refcnt;
    U32 sv_flags;
}
#define SVf_IOK 0x00100000
#define SVp_IOK 0x01000000
#define SVp_NOK 0x02000000

2332 int flags = sv->sv_flags;
2333 ...
2335 if((flags & SVf_IOK) ||
      ((flags & (SVp_IOK | SVp_NOK)) == SVp_IOK)){
2336 ...

```

B. Approach Overview

Based on the discussions above, we propose a novel approach to improve the integer signedness error detection using static data flow analysis. We at first identify *unharmsful integer signedness conversions* based on the data flow characteristics, and then exclude them from further instrumentations. Specifically, we first assume that all integer signedness conversions in program are un-trusted and each of them should be instrumented with security check to guarantee the runtime safety. Then we conduct static data flow analysis to mark unharmsful integer conversions in three aspects. 1) One taint-like analysis is employed to propagate the tag of ‘constant’ starting from constant values and safe library function calls (e.g., *uname()*, *gettimeofday()*). An integer conversion is marked as unharmsful if the source operand is tagged with ‘constant’; 2) An integer signedness conversion is also marked as unharmsful if it is protected by post-condition test; 3) Another data flow tracking

is deployed to compute an integer signedness conversion’s reachability to security-related operations. Note that this process is accomplished through determining whether it will encounter uncritical program locations or *SantzOp* before sinks on each path of this signedness conversion. If so, we mark this conversion as unharmsful. At last, security checks are inserted around the integer signedness conversions, which are not identified as unharmsful, so as to gain runtime protection, as existing instrumentation-based detectors [2][3][8][6] do.

III. DESIGN AND IMPLEMENTATION

Figure 1 illustrates the overall architecture of SignFlow. It performs static data flow analysis on the GCC intermediate representation—*gimple*—to identify unharmsful integer signedness conversions, and then instruments security checks for the remainder. At last, the inputs get further compilation and linking into binary. Specifically, SignFlow consists of four main components: 1) *constant type propagation analysis* is similar to taint-like analysis, aiming to identify all the integer signedness conversions under P_{cst} ; 2) *post-condition test identification* checks whether an integer signedness conversion is followed by post-condition test; 3) *reachability analysis* aims to compute the reachability of an integer signedness conversion to security-related operations by determining whether there exists uncritical program location or *SantzOp* along each path; 4) *instrumentation* part inserts security checks for integer signedness conversions except those, which have been identified as unharmsful by previous analyses.

Similar to IntFlow [6], our constant type propagation analysis is implemented via a taint-like analysis, i.e. setting the trusted source as ‘constant’, propagating this tag along the data flows, and marking the integer signedness conversion as unharmsful if the source operand is ‘constant’. Moreover, the process of post-condition test identification can also be easily implemented since the post-condition test particularly for signedness errors is of fixed patterns, as shown in Table I. Hence, in the following we will discuss more about reachability analysis and instrumentation.

A. Reachability Analysis

In this section, we analyze the data flows out from an integer signedness conversion to decide whether this conversion satisfies condition 1.2 or condition 1.3 (in Definition 1). If so, this integer signedness conversion is treated as unharmsful.

Whether an integer signedness conversion would be used at specific locations can be induced as a reachability problem. We

Algorithm 1 Reachability Analysis.

Input: Signedness conversion SC .
Output: $Sink, Unharm$;
1: $Sink, Unharm$ are initialized with 0;
2: **if** SC is unused afterward **then**
3: set $Unharm$;
4: **return** ;
5: **end if**
6:
7: **for** each $path$ of SC **do**
8: **for** each $node$ of $path$ **do**
9: **if** $node$ is uncritical site or SantzOp **then**
10: set $Unharm$;
11: **else if** $node$ is security-related operations **then**
12: set $Sink$; **break**;
13: **else if** $node$ is assignment, unary and binary op **then**
14: continue;
15: **else**
16: set $Sink$; **break**;
17: **end if**
18: **end for**
19: **if** $Sink$ **then**
20: clear $Unharm$; **break**;
21: **end if**
22: **end for**

compute the reachability of an integer signedness conversion to uncritical program locations and SantzOp to decide whether it satisfies condition 1.2, condition 1.3 or both. As shown in Algorithm 1, an integer signedness conversion can be treated as unarmful if 1) it is unused afterward (lines 2 to 5), 2) it ends up with uncritical program locations, or 3) it encounters SantzOp before security-related operations for each path. Note that security-related operations include *if*-statement, *while*-statement, array indexing and sensitive library routines such as *malloc()* and *memcpy()*. Lines 15 to 16 mean that our analysis is conservative for harmful signedness errors, i.e. an integer signedness conversion cannot be marked as unarmful if we are not certain of whether there will be sinks along some path.

B. Instrumentation

Through constant type propagation analysis, post-condition test identification and reachability analysis, a number of integer signedness conversions have been identified as unarmful. As the last step of SignFlow, security checks are inserted at the remaining signedness conversions for further runtime protection. We leverage pre-condition test [3][8], with the feature of testing whether misinterpretation will occur without actually performing the conversion. For instance, casting signed integer to unsigned type, i.e. `unsigned int a; signed b; a = (unsigned) b`, will cause signedness error if and only if the following expression is true: $(b < 0)$.

C. Implementation Details

We have implemented SignFlow for C/C++ programs based on GCC-4.5.0 [7]. Specifically, SignFlow is an optimization pass written in $\sim 4,000$ lines of C on *gimple*. *gimple* provides many interfaces for users to analyze the abstract syntax tree (AST), control flow graph (CFG) and call graph. Our constant

TABLE III. 7 HARMFUL INTEGER SIGNEDNESS BUGS IN REAL WORLD

CVE	Programs	Version	Sign Conv.	Sink
2008-1803	Rdesktop	1.5.0	$u \rightarrow s$	bound check
2009-3743	GhostScript	8.70	$s \rightarrow u$	memmove
2011-1471	PHP	5.3.6*	$s \rightarrow u$	bound check
2012-3368	Dtach	0.8	$s \rightarrow u$	bound check
2013-4927	Wireshark	1.10.0	$u \rightarrow s$	loop
2013-6489	Pidgin	2.10.11*	$s \rightarrow u$	malloc
2014-0045	Mumble	1.2.4	$s \rightarrow u$	malloc

type propagation analysis is accomplished by binding one tag ‘constant’ with each variable node and updating this tag with the traversal of each statement in AST. Our reachability analysis utilizes the propagation analysis engine in GCC, which is widely used by optimizations such as the copy propagation analysis and value range propagation analysis, to traverse each potential path of an integer signedness conversion. At last, security checks are inserted for those integer signedness conversions which are not marked as unarmful. The runtime handler is linked into the compilers’ output and takes actions when signedness errors are caught. It is worth noting that SignFlow works at *gimple* mainly because all implicit signedness castings are presented explicitly at this stage.

IV. EVALUATION

In this section, we present the results of our experimental evaluation using our prototype implementation of SignFlow, and compare the results with instrument-all techniques and IntFlow. All experiments were performed on an Intel Dual Core 2.4 GHz machine with 4GB memory. The OS is Linux-3.5.0. GCC was ran under `-O0` optimization level.

A. Detecting Harmful Integer Signedness Bugs

In order to evaluate the effectiveness of SignFlow in detecting and preventing harmful integer signedness bugs, we select 7 real-world signedness bugs published by CVE [9] as our test subjects, as shown in Table III¹. Columns 1 to 3 describe the CVE number, vulnerable software and version. Column 4 refers to signedness error site. That is the specific conversion, where signedness error occurs. $u \rightarrow s$ denotes casting unsigned integer to signed integer and $s \rightarrow u$ denotes casting signed integer to unsigned. Column 5 describes the security-related operation where the misinterpreted data is exploited.

The evaluation result is that *SignFlow successfully instrumented all the signedness error sites, i.e. SignFlow didn’t mark them as unarmful at the static data flow analysis phase*. To evaluate the runtime protection of SignFlow, we face the challenge that the corresponding signedness-error-inducing inputs are not available. We turn to extract the vulnerable conversion sites and their propagation paths from subject programs, and then execute the extracts with the self-designed signedness-error-inducing inputs. The result is *SignFlow reported warnings for all harmful signedness bugs*.

¹Detailed information about these 7 bugs can be referred at CVE website [9] via the corresponding CVE number. For CVE-2011-1471, the vulnerable version should be 5.3.5 and below. However, these versions of PHP cannot be compiled successfully under our experimental environment due to certain reason. Therefore, we choose 5.3.6 version instead and remove the patch manually. So it is with CVE-2013-6489.

TABLE IV. INTEGER SIGNEDNESS ERRORS REPORTED BY ALL, SignFlow_{cst} AND SignFlow

	#A	#S _c	#S	data flow pattern for each excluded error			
				P_{cst}	P_{uncrit}	P_{post}	P_{op}
400.perlbench	48	48	24			18	6
401.bzip2	15	14	7	1		4	4
445.gobmk	17	15	12	2		1	2
458.sjeng	3	3	0				3
462.libquantum	4	4	3		1		
464.h264ref	10	8	7	2			1
483.xalancbmk	9	7	6	2			1
Gzip	1	1	0				1
Dillo	5	5	4			1	
SWFTools	6	6	2			1	3
Total	118	111	65	7	1	25	21

Hence we gain confidence that SignFlow is suitable as a detection tool for real-world applications.

B. Reduction of False Positives

Reducing the number of false positives is the major goal of SignFlow , and this section quantifies how good SignFlow is in omitting unarmful signedness errors from the reported results by instrument-all techniques and IntFlow respectively. Here we implemented a prototype, named *ALL*, for instrument-all techniques by disabling our static data flow analysis, and a prototype, named SignFlow_{cst} , for IntFlow by only validating the constant type propagation analysis (i.e. disabling the post-condition test identification and reachability analysis).

We use SPECint 2006 benchmarks and 5 real-world applications as our testbed. We ran SPECint 2006 with the ‘ref’ input set. For SWFTools-0.9.1, we used the pdf2swf utility with the call-for-paper of SEKE’2015 as input; for Dillo-3.0.4.1, we visited its homepage and downloaded the source code; for Pidgin-2.10.11, we registered a new account, logged in and out; for Gzip-1.4, we compressed the archive gcc-4.5.0.tar; and for wget-1.6, we downloaded a 70MB file from a remote server. As each program is run with benign inputs, the reported integer signedness errors are all unarmful, i.e. false positives. We applied *ALL*, SignFlow_{cst} and SignFlow respectively on the testbed and calculated the reduction of false positives by SignFlow over *ALL* and SignFlow_{cst} . In addition, for each unarmful signedness error that SignFlow bypassed, we manually examined our static analysis results to find out which data flow pattern (as discussed in Section II) this signedness error belongs to. We report our findings in Table IV. For brevity, we only displayed the result of programs which have signedness errors.

Columns 2 to 4 show the number of integer signedness errors reported by *ALL*, SignFlow_{cst} and SignFlow respectively. Overall, SignFlow was able to suppress about 45% (53 out of 118) integer signedness errors reported by *ALL*, and this reduction is about 41% (46 out of 111) over SignFlow_{cst} . These were achieved due to the data flow characteristics of unarmful signedness errors used by SignFlow .

Columns 5 to 8 show the distributions of different data flow patterns, to which each excluded unarmful signedness errors belong. From the result we can observe that 7 out of 53 unarmful signedness errors identified by SignFlow are under P_{cst} while the other 47 are under P_{uncrit} , P_{post} or P_{op} . Hence as SignFlow considers whether the data after conversions could propagate to security-related operations or

TABLE V. CHECK DENSITY OF SignFlow

	#SC	#U	data flow pattern for each conv. in #U			
			P_{cst}	P_{uncrit}	P_{post}	P_{op}
400.perlbench	5955	2173	287	20	199	1684
401.bzip2	1105	110	67	4	4	35
429.mcf	72	8	8	0	0	0
445.gobmk	2374	706	639	9	1	57
456.hmmer	5589	2991	2927	8	1	53
458.esjeng	332	116	94	5	0	19
462.libquantum	312	185	148	7	0	29
464.h264ref	8274	2907	2808	0	5	84
471.omnetpp	1715	1151	1144	2	1	2
473.aster	372	16	16	0	0	0
483.xalancbmk	6163	681	387	1	28	236
Total	32263	10998	8525	56	239	2199

not, it can identify more unarmful signedness errors than SignFlow_{cst} , i.e. SignFlow improved the accuracy of detecting integer signedness errors a lot over IntFlow . Besides, the distributions of data flow patterns vary over different programs. This is mainly because the effectiveness in the reduction of reporting unarmful signedness errors is highly dependent on the nature of each program as well as on the level of the execution’s source coverage.

C. Performance

Check density refers to the ratio of the number of instrumented integer signedness conversions over all. Table V shows the experimental results of applying SignFlow to SPECint 2006 benchmarks. Column 2 shows the number of integer signedness conversions in *gimple*, and Column 3 indicates the number of integer signedness conversions identified as unarmful by SignFlow . In total, about 34% (10,998 out of 32,263) integer signedness conversions are marked as unarmful and excluded from instrumentation by SignFlow . In another word, the rest 66% signedness conversions are instrumented for runtime protection, i.e. the check density of SignFlow .

Columns 4 to 7 present the distributions of different data flow patterns, to which each unarmful integer signedness conversion belongs. Note that as some signedness conversion can be under several different data flow patterns, the value of Column 3 might be less than the sum of Columns 4 to 7. Form the result, we can see P_{cst} is the main type of unarmful integer signedness conversions. About 26.42% (8,528 out of 32,263) belongs to this pattern. It also denotes the number of signedness conversions excluded by IntFlow . Since SignFlow considers P_{uncrit} , P_{post} and P_{op} in addition, 2,494 more signedness conversions are identified as unarmful and excluded from instrumentation, compared to IntFlow .

Then We executed the instrumented program with the ‘ref’ input sets to test the overhead imposed by SignFlow . We ran each program 5 times and took the average value as the execution time. Note that the runtime handler is set as `nop` instruction when an integer signedness error is caught. We also have tested *ALL* and SignFlow_{cst} in the same way so as to illustrate the reduction of performance overhead that SignFlow gained. Compared to original benchmarks, the runtime overhead of *ALL*, SignFlow_{cst} and SignFlow are 6.19%, 5.35% and 2.70% respectively, which indicates that the performance loss of instrumentation-based techniques is quite low. On the other hand, SignFlow reduced the runtime overhead by

56.41% over ALL and 49.62% over SignFlow_{cst} respectively. Such reduction is achieved by our static data flow analysis, especially for that about 26.42% all signedness conversions are excluded from instrumentation as they originate from trusted source (i.e. under P_{cst}), and another 7.73% are excluded in the view of how they are used afterward (i.e. under P_{uncrit} , P_{post} or P_{op}).

D. Limitations

We propose to improve the accuracy of integer signedness error detection using the data flow characteristics. As an initial step along this direction, SignFlow has a number of limitations. *First*, SignFlow in current implementation identifies bitwise- and modulo operation as SantzOp by only checking whether b is a constant less than 0×80000000 . Precise integer range analysis is in need to identify more SantzOp. *Second*, SignFlow might fail to identify some data flow patterns due to the common challenges, such as pointer analysis and field-sensitiveness problem, faced by the static data flow analysis. *Third*, the scope of our static data flow analysis is limited to one single object file, as it is implemented as a compile-time optimization pass and not as a link-time optimization pass. This would also affect the detection accuracy of SignFlow. We are working on addressing these problems.

V. RELATED WORK

During the past years, great focus was placed upon dealing with integer signedness errors. Safe library functions are used to wrap integer signedness conversions by adding check code, such as IntegerLib [10] and Ranged Integer [11]. SmartFuzz [12] utilizes symbolic execution to generate test cases to invoke integer errors. The key challenges are to construct test cases of high code coverage and to deal with the path explosion problem when applied to large scale software.

Instrument-all techniques, such as RICH [2], IOC [3], RA [5] and AIR [4] inserts security checks for all integer signedness conversions for runtime protection. RICH provides formal specifications for integer semantics in C, and applies sub-type theory from type safe languages into C language. IOC has been integrated into LLVM compiler. The main drawback of instrument-all techniques is that they produce many false positives as they instrument security checks blindly. As shown in Section IV-B, SignFlow reduced about 45% false positives produced by ALL, a prototype of instrument-all technique.

IntFlow [6] aims to improve the accuracy of integer arithmetic errors. The difference from SignFlow lies in: 1) IntFlow focuses on excluding the false positives for *developer-intended undefined behavior*, including not only signedness errors, but also signed integer overflows, oversized shift and division by zero error; 2) IntFlow identifies unarmful undefined behaviors by checking whether they originate from trusted source, i.e. IntFlow only consider P_{cst} . In total, IntFlow achieves a reduction of 89% in false positives for all these undefined behavior; however it doesn't provide detailed statistics on its effectiveness particularly for integer signedness error. Therefore in order to compare IntFlow and SignFlow, we implemented SignFlow_{cst} as a prototype of IntFlow, and conducted a set of experiments on it. As SignFlow further considers how misinterpreted data is used afterward, i.e. the other three types of data flow

patterns for unarmful signedness errors, not limited to P_{cst} , it describes richer features of unarmful signedness error than IntFlow. Experimental results showed that SignFlow excluded 7.73% more integer signedness conversions from instrumentation than IntFlow (Section IV-C) and reduced 41% false positives that IntFlow produced (Section IV-B).

VI. CONCLUSION

To improve the accuracy of integer signedness error detection, in this paper we first defined the data flow patterns for unarmful signedness error from the view of where they originate and whether they can propagate to security-related operations, and further proposed and designed an instrumentation-based runtime integer signedness error detector, which could improve the precision a lot without sacrificing the completeness. A prototype, SignFlow is implemented as an extension of GCC. Experiments demonstrated that our tool can detect all harmful signedness bugs from our testbed while reducing 41% reports for unarmful ones. In our future work, we will study more features of unarmful signedness errors so as to further improve the detection accuracy. In addition, we will extend SignFlow to handle other types of vulnerabilities.

ACKNOWLEDGMENT

This work has been partly supported by National NSF of China under Grant No. 61170070, 61431008, 61321491; National Key Technology R&D Program of China under Grant No. 2012BAK26B01; the Program B for Outstanding PhD candidate of Nanjing University.

REFERENCES

- [1] National Vulnerability Database, "Mumble Opus Voice Packet Handling Remote Buffer Overflow," <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0045>.
- [2] D. Brumley, D. X. Song, T. cker Chiueh, R. Johnson, and H. Lin, "Rich: Automatically protecting against integer-based vulnerabilities," in *NDSS'07*, 2007.
- [3] W. Dietz, P. Li, J. Regehr, and V. S. Adve, "Understanding integer overflow in c/c++," in *ICSE'12*, 2012, pp. 760–770.
- [4] R. B. Dannenberg, W. Dormann, D. Keaton, R. C. Seacord, D. Svoboda, A. Volkovitsky, T. Wilson, and T. Plum, "As-if infinitely ranged integer model," in *ISSRE'10*, 2010, pp. 91–100.
- [5] R. E. Rodrigues, V. H. S. Campos, and F. M. Q. Pereira, "A fast and low-overhead technique to secure programs against integer overflows," in *CGO'13*, 2013, pp. 1–11.
- [6] M. Pomonis, T. Petsios, K. Jee, M. Polychronakis, and A. D. Keromytis, "Intflow: improving the accuracy of arithmetic error detection using information flow tracking," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 416–425.
- [7] "GCC, the GNU Compiler Collection," <https://gcc.gnu.org/>.
- [8] H. Sun, X. Zhang, C. Su, and Q. Zeng, "Efficient dynamic tracking technique for detecting integer-overflow-to-buffer-overflow vulnerability," in *AsiaCCS'2015*, 2015.
- [9] MITRE Corporation, "Common vulnerabilities and exposures," <http://cve.mitre.org/>.
- [10] CERT, "Integerlib, a secure integer library," 2006, <http://www.cert.org/secure-coding/IntegerLib.zip>.
- [11] J. Gennari, S. Hedrick, F. Long, J. Pincar, and R. C. Seacord, "Ranged integers for the c programming language," Carnegie Mellon University, Technical Note CMU/SEI-2007-TN-027, 2007.
- [12] D. Molnar, X. C. Li, and D. Wagner, "Dynamic test generation to find integer bugs in x86 binary linux programs," in *USENIX Security Symposium'09*, 2009, pp. 67–82.

Extracting More Object Usage Scenarios for API Protocol Mining

Deng Chen^{1, a}, Yanduo Zhang^{2, b}, Rongcun Wang^{3, c}, Binbin Qu^{4, d}, Jianping Ju^{5, e}, Wei Wei^{1, f}

¹ Industrial Robot Engineering Center, Wuhan Institute of Technology, Wuhan, P.R. China

² Hubei Provincial Key Laboratory of Intelligent Robot, Wuhan Institute of Technology, Wuhan, P.R. China

³ School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, P.R. China

⁴ School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, P.R. China

⁵ School of Electronic Information, Wuhan University, Wuhan, P.R. China

^a chendeng8899@hust.edu.cn

^b zhangyanduo@hotmail.com

^c rcwang@hust.edu.cn

^d bbqu@hust.edu.cn

^e gjdxjjp@whu.edu.cn

^f weiwei@huawei-elec.com

Abstract—Automatic protocol mining is a promising approach to infer precise and complete API protocols. However, the effect of the approach largely depends upon the quality of input object usage scenarios, in terms of noise and diversity. This paper aims to extract as many object usage scenarios as possible from object-oriented programs for automatic protocol mining. A large corpus of object usage scenarios can help with eliminating noise accurately and is likely to be diverse. Therefore, precise and complete protocols may be achieved. Given an object-oriented program p , generally, object usage scenarios that can be collected from a run of p is not more than the number of instances used in p . Relying on the inheritance relationship among classes, our technique can extract a maximum of n times more object usage scenarios from p , where n is the average inheritance depth of all object usage scenarios in p . In order to investigate the effect of our technique on mining protocols, we implement it in our previous prototype tool ISpecMiner and use the tool to mine protocols from several real-world applications. The experimental results show that our technique is promising to achieve complete and precise API protocols. In addition, protocols of classes that have not been used in programs can be also achieved, which is helpful for program documentation and understanding.

Keywords—object usage scenario; mining API protocol; object-oriented program; program verification

I. INTRODUCTION

Many application programming interfaces (APIs) impose protocols, that is, temporal constraints regarding the order of calls of API methods. For example, calling `peek()` on `java.util.Stack` without a preceding `push()` gives an `EmptyStackException`, and calling `next()` on `java.util.Iterator` without checking whether there is a next element with `hasNext()` can result in a

`NoSuchElementException`. API clients that violate such protocols do not obtain the desired behaviors and may even crash the program [1].

Automatic protocol mining [2]-[3] is a promising approach to infer precise and complete API protocols. These approaches first extract object usage scenarios from program applications statically or dynamically. Then, they take object usage scenarios as input and synthesize protocols based on sequential data mining techniques. However, the effect of these approaches largely depends upon the quality of input object usage scenarios: (1) noisy object usage scenarios will incur imprecision to mined protocols; and (2) in order to mine complete protocols, a set of diverse object usage scenarios is required.

Instead of improving the quality of input object usage scenarios directly, our work aims to extract as many object usage scenarios as possible from object-oriented programs for automatic protocol mining. Since a large corpus of object usage scenarios can compensate the inaccuracy caused by noise and is likely to be diverse [4], precise and complete protocols may be achieved. Generally, the number of object usage scenarios that can be collected from a run of an object-oriented program is less than or equal to the number of instances used in the program. Therefore, if a class is seldom used in a program, we will achieve insufficient object usage scenarios. Although feeding protocol miners more programs can mitigate the problem to some extent, much time overhead will be incurred.

Our technique is based on the following heuristic for object-oriented programs. Let c_1 and c_2 be two classes. If c_2 inherits from c_1 , c_2 will inherit the set of public methods (we omit other kinds of methods, e.g. protected methods and private methods, because API protocols always consider public methods of classes) M of c_1 as well as the temporal constraints regarding the order of calls of methods

(DOI reference number: 10.18293/SEKE2015-212)

in M . Or in other words, c_2 should not violate the temporal constraints regarding the order of calls of methods in M imposed by c_1 even if it overrides the inherited methods. Consequently, given an object usage scenario u of class c_2 , u should comply with the API protocols of c_2 as well as that of c_1 . Based on the above analysis, we derive an extra object usage scenario u' from u , which consists of methods inherited from c_1 . The extra object usage scenario is used to synthesize protocols of class c_1 . Theoretically, given an object-oriented program p , our technique can maximally extract n times more object usage scenarios from p than general approaches, where n is the average inheritance depth of all object usage scenarios in p . To investigate our technique's feasibility and effectiveness, we implemented it in our previous dynamic program specification mining tool ISpecMiner and used the tool to conduct experiments. Results of the experiments show that our technique is promising to achieve complete and precise protocols.

The contributions of this paper are:

- A technique that can extract more object usage scenarios from object-oriented programs than existing approaches.
- A formal discussion about how many more object usage scenarios can be collected by our technique.
- Investigation of the effect of our technique on mining API protocols.

II. PRELIMINARY

In this section, we present some preliminaries that our work is based on.

DEFINITION 1 (Object Usage Scenario). Let c be a class. An *Object Usage Scenario (OUS)* of c is a method call sequence, all methods in the sequence are called on a same instance of c . Assume that s is an instance of class c . We use $ous(s)$ to denote an object usage scenario of c , each method of which is called on s . Furthermore, we use $OUS(c)$ to denote the set of object usage scenarios of class c , each element of which is an object usage scenario of an instance of c .

Consider the Java program illustrated in Figure 1, which makes use of classes `FileInputStream` and `FileOutputStream`. Let's assume that the loop iterates only once. We will achieve the following OUSs regarding

```

1) FileInputStream fis = new FileInputStream("filepath");
2) FileOutputStream fos = new FileOutputStream("filepath");
3) byte[] buffer = new byte[1024];
4) int count = 0;
5) while ((count = fis.read(buffer)) != -1)
6) {
7)   fos.write(buffer, 0, count);
8) }
9) fis.close();
10) fos.close();

```

Figure 1. Java program example.

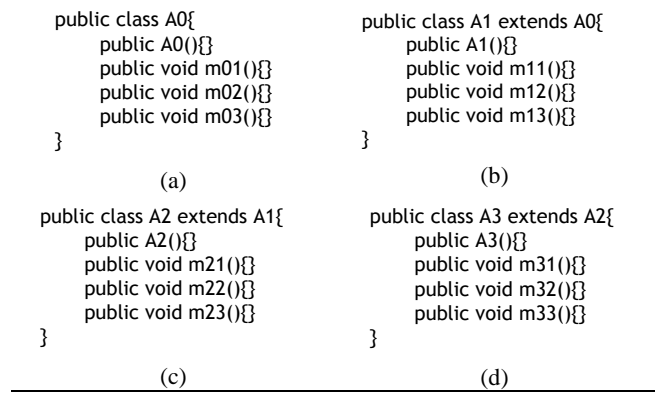


Figure 2. Program examples of inheritance relationship.

instance `fis` and `fos` respectively:

- $ous(fis)$: `<FileInputStream(), read(), close(>`
- $ous(fos)$: `<FileOutputStream(), write(), close(>`

As we can see, an OUS u of class c represents a use case about how client programs should use methods of c . Therefore, through extracting common patterns from a set of OUSs of c , we can infer protocols of class c . Additionally, given an object-oriented program p , OUSs that can be collected from a run of p is less than or equal to the number of instances used in p . In Section 3, we will show that our technique can extract multiple times more OUSs than general approaches.

III. OUR TECHNIQUE

In object-oriented programs, classes have inheritance relationships among them. Consider the Java programs illustrated in Figure 2, the four classes `A0`, `A1`, `A2` and `A3` have the following inheritance relationships: class `A1` inherits from `A0`, class `A2` inherits from `A1` and class `A3` inherits from `A2`. Since a subclass will inherit public methods of its superclasses, the above classes have the public methods listed in Table I. Our approach is based on the following heuristic.

HEURISTIC 1. Let r be a class. M is the set of public methods of r . We use ρ_r^M to denote the API protocol regarding methods in M imposed by class r . Assume that c is a subclass of r , which should inherit all methods in M from r . We have $\rho_c^M \circ \rho_r^M$, where \circ represents that

TABLE I. PUBLIC METHODS OF CLASSES WITH INHERITANCE RELATIONSHIPS. METHODS INHERITED FROM SUPERCLASSES ARE IN ITALIC.

Class	Public methods
A0	m01, m02, m03
A1	<i>m01</i> , <i>m02</i> , <i>m03</i> , m11, m12, m13
A2	<i>m01</i> , <i>m02</i> , <i>m03</i> , <i>m03</i> , <i>m11</i> , <i>m12</i> , <i>m13</i> , m21, m22, m23
A3	<i>m01</i> , <i>m02</i> , <i>m03</i> , <i>m11</i> , <i>m12</i> , <i>m13</i> , <i>m21</i> , <i>m22</i> , <i>m23</i> , m31, m32, m33

protocol ρ_c^M is *equivalent to or stricter than* ρ_r^M . Or in other words, the implementation of a subclass should not violate API protocol imposed by its superclasses.

We make the heuristic based on the following literature: From the perspective of data abstraction and hierarchy [5], a *subtype* is one whose objects provide all the behavior of objects of another type (the *supertype*) plus something extra. Furthermore, we have the following substitution property: If for each object o_1 of type S there is an object o_2 of type T such that for all programs P defined in terms of T , the behavior of P is unchanged when o_1 is substituted for o_2 , then S is a subtype of T [6]. The above data abstraction and hierarchy principles are supported by linguistic mechanisms in many object-oriented programming languages, such as Simula 67, CLU, Smalltalk and Java. A typical case in Java language is the exception handling mechanism: In Java programs, a method can incur exceptions through the *throw* statement. What is interesting is that, when a method is reimplemented in a subclass, the exceptions thrown in superclasses should be inherited. For example, assume that E is the set of exceptions thrown by method **m01** defined in class **A0** (shown in Figure 2). When we overwrite method **m01** in class **A1**, all exceptions in E should also be thrown, that is, a subclass should not violate restrictions on exceptions imposed by its superclasses. Our case is similar to the exception handling mechanism in Java language. What is different is that, we consider constraints regarding order of calls of methods rather than exceptions. Whatever, according to the data abstraction and hierarchy principle, we have the following conclusion: a subtype should not violate the API protocol imposed by its supertype regarding inherited methods, otherwise the substitution property cannot be satisfied. What may occur is that the subtype has a stricter restriction than supertypes on the order of calls of inherited methods.

According to Heuristic 1, given an OUS u of class c , if u can be accepted by the API protocol of c , u should also comply with the protocol of the superclasses of c regarding inherited methods, because the protocol imposed by c is equivalent to or stricter than that imposed by superclasses. Consequently, we can derive an extra OUS from u , which consists of calls of inherited methods. We call the derived OUS *inherited sub-OUS*, which can be used to mine protocols of the superclasses of c . Formally, we give the following definitions.

DEFINITION 2 (Sub-OUS). Given an OUS u , OUS u' is a *sub-OUS* of u , if it satisfies the following requirements.

- Each method call in u' is included in u .
- Let m_1 and m_2 be two method calls in u' , the temporal relationship between m_1 and m_2 should be consistent in u' and u , that is, if m_1 precedes m_2 in u' , m_1 should also precede m_2 in u .

DEFINITION 3 (Inherited Sub-OUS). Given an OUS u of class c which consists of calls of inherited methods and those defined in c itself, u' is a sub-OUS of u , each element of which is a call of method inherited from a superclass c' of c , we call u' an *inherited sub-OUS* of u from c' and denote it by *isub-OUS*(u, c').

For example, given an OUS u : <m01, m11, m03, m02, m12, m31, m32, m20, m23, m33> of class **A3**, according to Definition 3, we can derive the following inherited sub-OUSs.

- *isub-OUS*($u, A0$): <m01, m03, m02>
- *isub-OUS*($u, A1$): <m01, m11, m03, m02, m12>
- *isub-OUS*($u, A2$): <m01, m11, m03, m02, m12, m20, m23>

As we can see, the above inherited sub-OUSs are sub-OUS of u . In addition, they consist of methods inherited from superclass **A0**, **A1** and **A2** respectively. Based on Heuristic 1, the inherited sub-OUSs should satisfy the API protocols of class **A0**, **A1** and **A2** respectively. Consequently, aside from OUS u which can be used to mine protocol of class **A3**, we will achieve three additional OUSs.

In addition, given an OUS u of class c , the number of inherited sub-OUSs of u is less than or equal to the inheritance depth of c . On the other hand, each instance in an object-oriented program will generate an OUS. Therefore, given an object-oriented program, the extra OUS (inherited sub-OUS) collected by our technique is maximally n times the number of instances defined in the program, where n is the average inheritance depth of all OUSs (excluding inherited sub-OUSs) in the program.

IV. FORMAL ANALYSIS OF THE EFFECT OF OUR TECHNIQUE

In this subsection, we analyze the effect of our technique formally.

Let's assume that p is a Java program. It subsumes the following OUSs u_1, u_2, \dots, u_m , which have an inheritance depth of d_1, d_2, \dots, d_m , respectively. Given an OUS u of class c , we call inheritance depth of c the inheritance depth of u . It must be noted that the number of inherited sub-OUSs of u is less than or equal to its inheritance depth. Therefore, the maximum total number of additional OUSs (inherited sub-OUSs) that can be collected by our technique from p is $d_1 + d_2 + \dots + d_m$. Let \bar{d} be the average inheritance depth of all OUSs (excluding inherited sub-OUSs) included in p . We have $d_1 + d_2 + \dots + d_m = m \times \bar{d}$.

Based on the above analysis, we reach the conclusion that our technique can extract maximally n times more OUSs from an object-oriented program, where n is the average inheritance depth of OUSs (excluding inherited sub-OUSs) in the program.

TABLE II. SUBJECT PROGRAMS. KLoC: THOUSANDS OF LINES OF CODE.

Subject	Version	Description	KLoC
FreeMind	0.9	Mind-mapping software	22
RapidMiner	5.3	Environment for machine learning and data mining	513
SQuirreL SQL Client	3.4	Java SQL client	253
OpenProj	1.4	Project management software	120

V. EXPERIMENTS

To evaluate our technique, we implemented it in our previous prototype tool *ISpecMiner* and used the tool to conduct experiments. In this section, we first introduce *ISpecMiner*. Then, we present subjects that are used in our evaluation. Finally, we compared API protocols achieved under our technique and general approaches.

A. Prototype Tool *ISpecMiner*

ISpecMiner [7] is a dynamic program specification mining tool developed based on Java 1.6. It leverages *Java agent* [8] technique as well as *Javassist* [9], [10] to extract OUSs from Java application programs dynamically, and then infers class temporal specifications (API protocols). The most distinguishing characteristic of *ISpecMiner* is that it describes program specifications using a probabilistic model extended from Markov chain. Probabilistic models have an inherent ability to tolerate noises. Furthermore, since *ISpecMiner* learns program specifications in an online mode, mined specifications can be evolved persistently. As a result, more universal program specifications can be achieved.

The number of OUSs that *ISpecMiner* extracts from a Java program is near the number of instances of classes used in the program. In this work, to prepare *ISpecMiner* for our experiments, we implemented our technique in it. In the remainder of this section, we denote original *ISpecMiner* and *ISpecMiner* with our technique by *ISpecMiner-I* and *ISpecMiner-II* respectively. The latest version of *ISpecMiner* can be obtained at the URL <http://www.ispecminer.com>.

B. Subjects

The subjects used in our experiment are shown in Table II, which are real-world Java programs. These programs are selected based on the following criteria:

- Open source software. Though *ISpecMiner* is a dynamic specification mining tool and source code is not necessary, it is helpful for us to figure out problems encountered in experiments and validate results.

TABLE III. INSTRUMENTED CLASSES

	Instrumented Class	Inheritance Depth
1	java.io.PushbackInputStream	2
2	java.io.FileInputStream	1
3	java.io.FileOutputStream	1
4	java.io.BufferedReader	1
5	java.io.BufferedWriter	1
6	java.io.DataInputStream	2
7	java.io.DataOutputStream	2
8	java.io.FileReader	2
9	java.io.FileWriter	2
10	java.io.BufferedInputStream	2
11	java.io.PrintWriter	1

- Large-scaled software. Large-scaled software contains a large number of OUSs, which is helpful for our comparison test.
- Applications coming from various domains. Applications from various areas may avoid the biases introduced in our experiments.

C. Investigation of API protocols

In order to investigate the effect of our method, we used *ISpecMiner-I* and *ISpecMiner-II* to mine API protocols from several real-world Java programs respectively, and then compared the achieved protocols. The subject programs are shown in Table II, each of which was run with manual input data. We configured *ISpecMiner-I* and *ISpecMiner-II* to instrument classes illustrated in Table III and their superclasses. The reasons we selected these classes are as follows: (1) they are commonly used in various kinds of Java programs; and (2) they have an inheritance depth greater than zero. Thus, an OUS of these classes will derive at least one inherited sub-OUS. It is worth noting that we exclude the common superclass *java.lang.Object* of all classes in Java when counting inheritance depth. Take for example, class *java.io.FileInputStream*, which has two superclasses *java.io.InputStream* and *java.lang.Object*. According to our counting method, it has an inheritance depth of one.

Experimental results are illustrated in Figure 3. We present the API protocols of class *InputStreamReader*, *InputStreamWriter* and *FilterInputStream* sequentially from the first row to the last row. At each row, the left protocol is mined by *ISpecMiner-I* and the right one is mined by *ISpecMiner-II*. The protocols are described using an extended Markov model MCF, where states and transitions represent methods and temporal relationships between methods respectively. Details about MCF please refer to [7]. From Figure 3, we can see that protocols in the right column have more states or transitions than those in the left column. For example, the API protocol of class *InputStreamReader* mined by *ISpecMiner-II* has one more state and four more transitions than that mined by

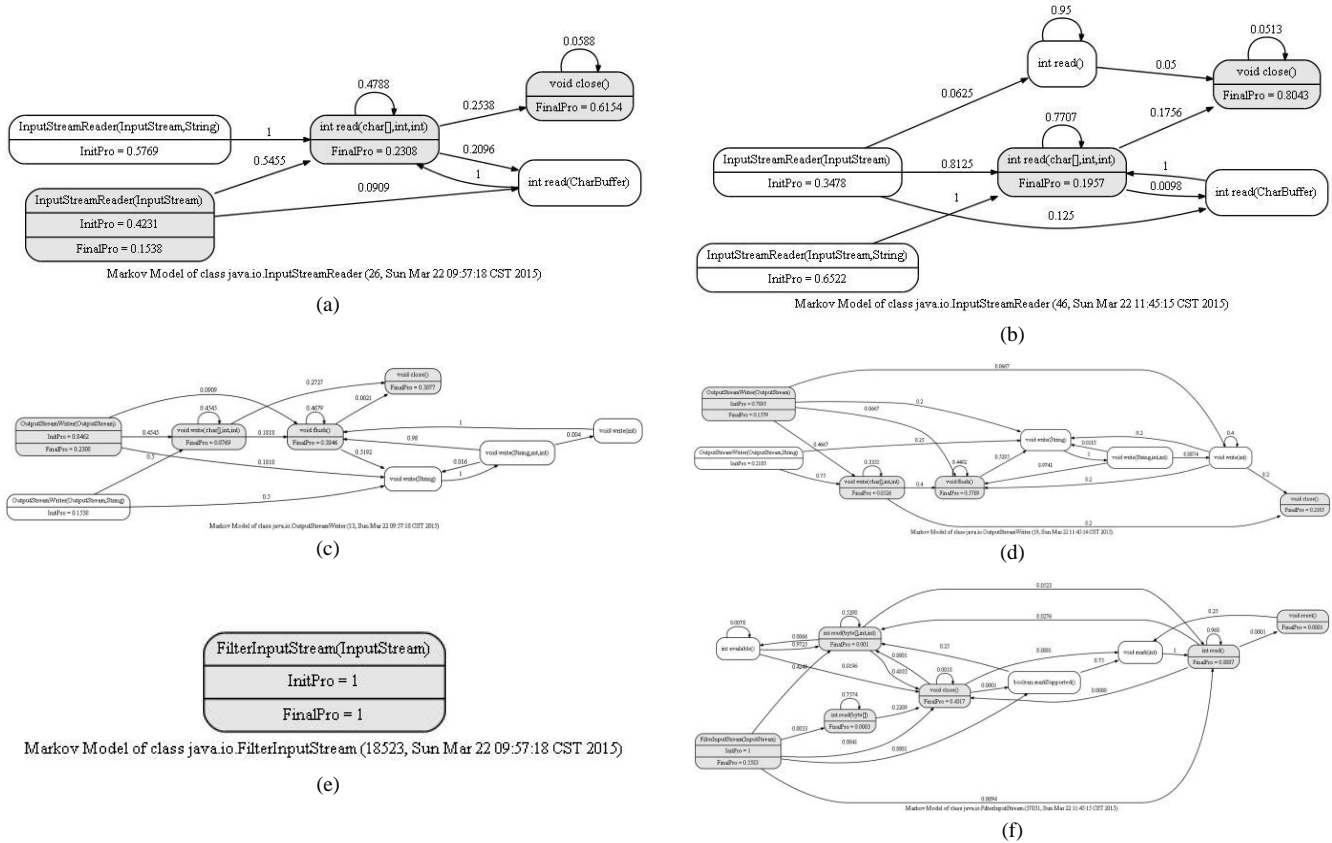


Figure 3. Part of API protocols mined in our experiment.

ISpecMiner-I. As to the protocols of class `OutputStreamWriter` shown in the second row, although they have the same number of states, the right one has many more transitions than the left one. By manual inspection, we found that the additional states and transitions are consistent with JDK documentations. Since the number of states and transitions reflects the comprehensiveness of protocols to some extent, we have the conclusion that our technique is helpful for mining comprehensive protocols.

On the other hand, our approach can impact the probabilities attached with states and transitions: probabilities of normal behaviors will be enhanced and that of abnormal behaviors will be suppressed. For example, the final probability (`FinalPro`) of state `close()` in protocol of class `InputStreamReader` shown in the first row is increased from 0.6154 to 0.8043 and that of state `read(char[],int,int)` is decreased from 0.2308 to 0.1957. Since the difference between normal and abnormal behaviors is enlarged, noisy states and transitions can be eliminated accurately when transforming probabilistic models to deterministic models using a probability threshold.

Additionally, our technique can achieve protocols that cannot be mined by general approaches. Take for example, the superclass `FilterInputStream` of `DataInputStream`

and `BufferedInputStream`. Since the class has not been covered during the run of subject programs, a protocol shown in Figure 3 (e) with only constructor method was achieved by ISpecMiner-I (the constructor method of class `FilterInputStream` may be called by constructor method of its subclasses). In contrast, ISpecMiner-II generated a more complete protocol illustrated in Figure 3 (f), because our method can derive inherited sub-OUSs. It seems that protocols as `FilterInputStream` are useless for program validation, because they are seldom used in application programs. However, there still exist many superclasses which are frequently used in programs, such as `InputStreamReader` and `OutputStreamWriter`. Since they have been used in subject programs, we can achieve relative complete protocols based on general approaches as shown in Figure 3 (a) and (c). Even if some classes will be never used in programs (such as abstract classes), their protocols may be useful in program documentation and understanding. For example, we can validate the design of an abstract class based on mined protocols.

D. Related Work

Many researchers have paid significant efforts in mining API protocols. For instance, Wasylkowski et al. [11] proposed to mine object usage models from Java bytecode

and a tool JADET was developed. Lorenzoli et al. [15] modeled API protocols using EFSM which extends from FSM. Alur et al. [16] synthesized FSA model of API protocols using L* learning algorithms combined with model checking and abstract interpretation techniques. Since FSA is a kind of deterministic model with inability to tolerate noise, many researchers proposed to mine API protocols based on probabilistic models. For example, Ammons et al. [17] proposed to mine protocols among application programming interfaces (API) or abstract data types (ADT) based on probabilistic finite state automaton (PFSA). Chen et al. [7] proposed to mine class temporal specifications based on an extended Markov model. Whatever techniques, the quality of input OUSs is important for mining precise and complete protocols. However, little attention has been paid in this area. In this paper, we proposed an approach to collect as many OUSs as possible for automatic protocol mining. A large repository of OUSs can complement the inaccuracy caused by noises and is likely to be diverse. Currently, a common approach to collect more OUSs is feeding protocol miners more application programs, which will incur significant time overhead. Different from that, our technique can extract more OUSs from a single application program.

VI. CONCLUSIONS

Automatic protocol mining is a promising approach to infer precise and complete API protocols. Many researchers have paid significant efforts in this area. However, little attention has been paid on collecting high quality OUSs. In this paper, we proposed an approach to collect more OUSs for API protocol mining. Our technique is based on the inheritance relationship among classes. Given an object-oriented program p , theoretically, n times more OUSs can be extracted by our technique from p than general approaches, where n is the average inheritance depth of all OUSs in p . In the Experimental Section, we investigated the effect of our approach on mined API protocols and found that our technique is promising to achieve complete and precise protocols. Additionally, our technique can mine protocols even if the corresponding classes have not been covered during the run of application programs. Although these protocols may be useless for program validation, they can be used for program documentation and understanding.

ACKNOWLEDGMENT

Supported by Natural Science Foundation of Hubei Province (No. 2014CFB1006).

REFERENCES

- [1] Pradel, M. and Gross, T. R. Leveraging test generation and specification mining for automated bug detection without false positives. In *ICSE'12: Proceedings of the 34th International Conference on Software Engineering*. Zurich, Switzerland, 2012, 288-298.
- [2] Ramanathan, M. K., Grama, A., et al. Static specification inference using predicate mining. *SIGPLAN Not.* 2007, 42(6), 123-134.
- [3] Shoham, S., Eran, Y., et al. Static specification mining using automata-based abstractions. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis*. United Kingdom: ACM, London, 2007.
- [4] Engler, D., Chen, D., et al. Bugs as deviant behavior: a general approach to inferring errors in systems code. *SIGOPS Oper. Syst. Rev.* 2001, 35(5), 57-72.
- [5] Liskov, B. Data abstraction and hierarchy. *SIGPLAN Not.* 1987, 23(5), 17-34.
- [6] Bruce, K.B. and Wegner, P. An algebraic model of subtypes in object-oriented languages. *SIGPLAN Not.* 1986, 21(10), 163-172.
- [7] Chen, D., Huang, R., et al. Mining class temporal specification dynamically based on extended Markov model. *International Journal of Software Engineering and Knowledge Engineering*. 2013, in press.
- [8] Caserta, P. and Zendra, O. JBInsTrace: a tracer of Java and JRE classes at basic-block granularity by dynamically instrumenting bytecode. *Science of Computer Programming*, 2014, 79 (SI), 116-125.
- [9] Tatsubori, M., Sasaki, T., et al. A bytecode translator for distributed execution of "legacy" Java software. In *Proceedings of the 15th European Conference on Object-Oriented Programming*. Springer-Verlag, 2001.
- [10] Javassist, 2013. <http://en.wikipedia.org/wiki/Javassist>.
- [11] Wasylkowski, A. Mining object usage models. In *Companion to the Proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society, 2007.
- [12] Wasylkowski, A., Zeller, A., et al. Detecting object usage anomalies. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Croatia: ACM, Dubrovnik, 2007.
- [13] JADET, 2014. <http://www.st.cs.uni-saarland.de/models/jadet/>.
- [14] Dallmeier, V., Lindig, C., et al. Mining object behavior with ADABU. In *Proceedings of the 2006 International Workshop on Dynamic Systems Analysis*. China: ACM, Shanghai, 2006.
- [15] Lorenzoli, D., Mariani, L., et al. Automatic generation of software behavioral models. In *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*. Germany: ACM, Leipzig, 2008.
- [16] Alur, R., Cerny, P., et al. Synthesis of interface specifications for Java classes. *SIGPLAN Not.* 2005, 40 (1), 98-109.
- [17] Ammons, G. and Bodik, R., et al. Mining specifications. *SIGPLAN Not.* 2002, 37 (1), 4-16.

NeoIDL: A Domain-Specific Language for Specifying REST Services

Rodrigo Bonifácio*, Thiago Mael de Castro†, Ricardo Fernandes†, Alisson Palmeira†, and Uirá Kulesza‡

* Departamento de Ciência da Computação, Universidade de Brasília, Brazil

† Centro de Desenvolvimento de Sistemas, Exército Brasileiro, Brazil

‡ Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte, Brazil

Abstract—Service-oriented computing has emerged as an effective approach for integrating business (and systems) that might spread throughout different organizations. A service is a unit of logic modularization that hides implementation details using well-defined contracts. However, existing languages for contract specification in this domain present several limitations. For instance, both WSDL and Swagger use language-independent data formats (XML and JSON) that are not suitable for specifying contracts and often lead to heavyweight specifications. Interface description languages, such as CORBA IDL and Apache Thrift, solve this issue by providing specific languages for contract specifications. Nevertheless, these languages do not target to the REST architectural style and lack support for language extensibility. In this paper we present the design and implementation of NeoIDL, an extensible domain specific language and program generator for writing REST based contracts that are further translated into service’s implementations. We also describe an evaluation that suggests the rapid return on investment with respect to the design and development of NeoIDL¹.

I. INTRODUCTION

Service-oriented computing (SOC) [6] is a consolidated approach that enables the development of low coupling systems, which are able to communicate to each other even across different domains. Thanks to the use of open standards and protocols (such as HTTP and HTTPS) in SOC, service orchestration enables the automation of business processes among different corporations. A service is defined as a unit of logic modularization [6] that hides implementation details and adheres to a contract, usually described using a specification language (for example WSDL [3], WADL [8], Swagger [18], Apache Thrift [17] or CORBA [13]).

There is a recent trend to shift the implementation of services using the set of W3C specifications for service-oriented computing (such as SOAP and WSDL) to a lightweight approach based on the REpresentational State Transfer (REST). REST is a stateless, client-server architectural style that is being used for service-oriented computing [7]. Although REST still lacks an agreement about a language for specifying contracts, Erl et al. [5] suggest that a REST contract should at least comprise a resource identification, a protocol method, and a media type. Currently, the existing approaches for specifying contracts in REST present some limitations. For instance, Swagger specifications [18] are written in JSON (Java Script Object Notation), a general purpose notation for data representation that often leads to lengthy contracts. Swagger

also does not provide any language construct for services and data type reuse. Apache Thrift provides a specification language more clear and concise, though its language is also limited with respect to both modularity and reuse, since it is not possible to specialize user defined data types (as it is possible using CORBA IDLs [13]). Furthermore, the Apache Thrift language does not present any means to extend the language used for specifying contracts.

In this paper we describe a new language— NeoIDL— for specifying REST services with their respective contracts and an extensible program generator that translates NeoIDL specifications into source code. Besides describing REST contracts in terms of resources, methods, and media types, NeoIDL specifications also include the definition of the data types used in the visible interface of a service. We considered the following requirements when designing NeoIDL. First, the language should be concise and easy to learn and understand. Second, the language should present a well-defined type system and support single inheritance of user defined data types. In addition, developers using NeoIDL should be able to specify concepts related to the *REST architectural style for service-oriented computing* [7], in order to simplify the translation of a NeoIDL specification into basic components tailored to that architectural style. Finally, both NeoIDL and the program generator should be extensible. For that reason, we designed NeoIDL to support extensibility through annotations; whereas the extensibility of the program generator relies on a pluggable architecture that uses high-order functions and some facilities present on the Glasgow Haskell Compiler (GHC) [12]. In summary, the contributions of this paper are twofold.

- We present the design and development of NeoIDL, a novel specification language for service-oriented computing that conforms to the aforementioned requirements (Section II).
- We present the implementation details of an extensible program generator written in Haskell (Section III). This contribution addresses the issue of building extensible architectures in a pure, statically typed functional language— a challenging that has not been completely discussed in the literature.

In Section IV we discuss the extensibility mechanisms and the return on investment of NeoIDL. Section V relates our contributions with existing research work available in the literature. Finally, Section VI presents final remarks and future directions of NeoIDL.

¹DOI reference number: 10.18293/SEKE2015-218

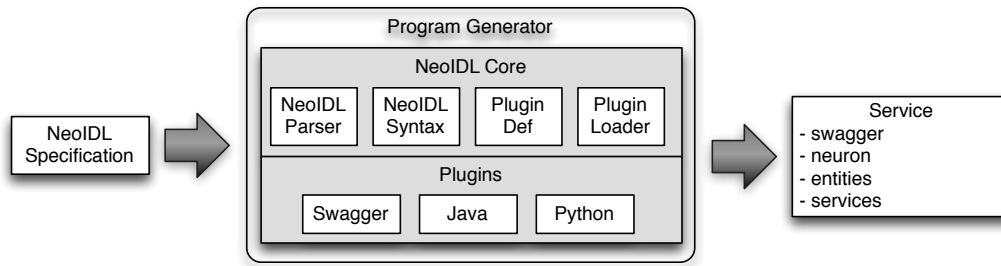


Fig. 1. Architecture of NeoIDL program generator

II. NEOIDL DESIGN

In this section we first present an overview of our approach (Section II-A), which consists of a specification language and a program generator. Then, in Section II-B, we detail the principal constructs of NeoIDL and illustrate some examples of service specifications.

A. Approach Overview

NeoIDL has been developed to enable the specification of REST services and to allow the code generation of the implementation of those services for specific platforms. It aims to simplify the development of services, by generating code from a service specification. Figure 1 illustrates the main components of our approach, which consists of: (i) a domain-specific language (NeoIDL) for specifying REST services with their respective contracts; and (ii) a program generator that enables the code generation of REST services in different platforms. The NeoIDL generator is structured as a set of core modules, which are responsible for the parsing, syntax definition, and processing of NeoIDL specification; as well as modules for the definition and management of NeoIDL plugins. Each NeoIDL plugin defines specific extensions for the code generator that enables the generation of REST services for different platforms or programming languages.

The current implementation of NeoIDL has been already used to enable the generation of services for the NeoCortex platform, a proprietary framework used by the Brazilian Army. NeoCortex is a service oriented framework based on REST that has been developed using NodeJS— a cross-platform runtime environment for server-side and networking applications. NeoCortex is a polyglot framework that supports the deployment of services written in different languages (such as Python and Java) and addresses high responsiveness requirements using reactive and asynchronous programming techniques. Each NeoCortex service must provide a contract and a *front-controller*— which delegates a service request to the corresponding implementation. Even simple NeoCortex services require different components that implement business logic and other concerns, such as concurrency and persistence. In summary, a typical NeoCortex service comprises several components, such as:

- The *synapse* component exposes the service API using a Swagger based JSON specification, which provides an useful interface for testing a service.

- The *neuron* component implements the necessary behavior for initializing and stopping a service, as well it is responsible for mapping a requested URL pattern into a specific resource class.
- User defined data types are represented as domain classes, either in Python or Java. In the cases where it is necessary to persist a data type on a database, a database mapping is also necessary within a service.

In the context of NeoCortex, we translate NeoIDL specifications into Swagger specifications and other software components for different programming languages— to fulfill the polyglot requirement of NeoCortex. This requirement motivated us to implement the program generator of NeoIDL as a pluggable architecture (see Figure 1)— so that we are able to evolve the code generation support in a modular way. For instance, implementing a C++ program generator from NeoIDL specifications does not require any change in the existing code of the program generator. It is only necessary to implement a new NeoIDL plugin.

B. NeoIDL Language

NeoIDL simplifies service specifications by means of (a) mechanisms for modularizing and inheriting user defined data types, and (b) a concise syntax that is quite similar to the *interface description languages* of Apache Thrift and CORBA. A NeoIDL specification might be split into modules, where each module contains several definitions. In essence, a NeoIDL definition might be either a data type (using the *entity* construct) or a service describing operations that might be reached by a given pair (URI, HTTP method). Figures 2 and 3 present two NeoIDL modules: (i) the data-oriented *MessageData* module; and the service-oriented *Message* module.²

The *MessageData* module (Figure 2) declares an enumeration (*MessageType*), which states the two valid types of messages (a message must be either a *message sent* or a *message received*); and a data type (*Message*), which details the expected structure of a message. We use a *convention over configuration approach*, assuming that all attributes of a user defined data type are mandatory, though it is possible to specify an attribute as being optional using the syntax `<Type> <Ident> = 0;`, as exemplified by the *subject* field of the *Message* data type.

²The NeoIDL grammar could be found at <http://goo.gl/p8eZky>

```

1 module MessageData {
2   enum MessageType { Received, Sent };
3
4   entity Message {
5     string id;
6     string from;
7     string to;
8     string subject = 0;
9     string content;
10    MessageType type;
11  };
12 }

```

Fig. 2. Message data type specified in NeoIDL

The `Message` module of Figure 3 specifies one service resource (`sentbox`). As explained, we send requests for the methods of a given resource using a specific path. In the example, the `sentbox` resource’s methods are available from the relative path `/messages/sent`. This resource declares two operations: one `POST` method that might be used for sending messages and one `GET` method that might be used for listing all messages sent from a given sequential number.

Also according to our *convention over configuration* approach, we assume that the arguments of `POST` and `PUT` operations are sent in the request body, whereas arguments of `GET` operations are either sent enclosed with the request URL or enclosed with the URL path (in a similar way as `DELETE` operations). We are able to change these conventions by using specific annotations attached to an operation parameter. In these examples, conventions are used to reduce the size of services’ specifications.

```

1 module Message {
2   import MessageData;
3
4   resource sentbox {
5     path = "/messages/sent";
6     @post void sendMessage(Message message);
7     @get [Message] listMessages(string seq);
8   };
9 }

```

Fig. 3. Sent message service specification in NeoIDL

To support language extensibility, NeoIDL specifications can be augmented through annotations. The main reason for introducing annotations in NeoIDL was the possibility to extend the semantics of a specification without the need to change the concrete syntax of NeoIDL. For instance, suppose that we want to express security policies for a service resource. A developer could change the concrete syntax of NeoIDL for this purpose, defining new language constructs for specifying the authentication method (based on tokens or user passwords), the cryptographic algorithm used in the resource request and response, and the role-based permissions to the resource capabilities. However, changing the concrete syntax to allow the specification of unanticipated properties of a resource often breaks the code of the program generator.

Instead, using annotations, developers might extend the language within NeoIDL specifications. Therefore, apart from

```

1 module Agente {
2
3   entity Agent {
4     ...
5   };
6
7   annotation SecurityPolicy for resource {
8     string method;
9     string algorithm;
10    string role;
11  };
12
13  @SecurityPolicy(method = "basic",
14                 algorithm="AES",
15                 role = "admin");
16  resource agent {
17    path = "/agent";
18    @post void persistAgent(Agent agent);
19  };
20 }

```

Fig. 4. NeoIDL specification using annotations

the NeoIDL definitions discussed before, it is also possible to define new annotations that might be attached to the fundamental constructs of NeoIDL (i.e. `module`, `enum`, `entity`, and `resource`). Each annotation consists of a name, a target element that indicates the NeoIDL constructs the annotation might be attached to, and a list of properties. When transforming a specification, the list of annotations attached to a NeoIDL element is available to the plugins, which could consider the additional semantics during the program generation. Figure 4 presents a NeoIDL example that attaches an user defined annotation (`SecurityPolicy`) to specify security policies on the `agent` resource. In the example, using the `SecurityPolicy` annotation we specify that the operations of the `agent` resource (i) must use a basic authentication mechanism, (ii) the arguments and return values must be encoded using the AES algorithm, and (iii) only authenticated users having the *admin* role are authorized to request the resources.

We end this section highlighting that the design of NeoIDL comprises a domain specific language (DSL) for specifying services APIs in a REST based environment and an extensible program generator that might evolve to generate code to different platforms and programming languages. Next section presents some details about the NeoIDL implementation, which uses Haskell as programming language—a well known language for building (embedded) DSLs [9].

III. NEOIDL IMPLEMENTATION

As shown in Figure 1, the implementation of NeoIDL consists of a core (split into several Haskell modules) and several plugins, one for each target language (such as Swagger, Python, or Java). The core module includes a tiny application that loads plugins definition and processes the program arguments, which specify the input NeoIDL file, the output directory, and the languages that should be generated code from the input file. Moreover, the core module contains a parser³ and a type checker for NeoIDL specifications.

³We have developed the parser for NeoIDL using `BNFConverter` [16]

In the remaining of this section we present details about the implementation of two NeoIDL Haskell modules: `PluginDef` and `PluginLoader`. The first states the organization of a NeoIDL plugin and the second is responsible for loading all available plugins. The details here are particularly useful for those who want to develop extensible architectures using Haskell.

A. `PluginDef` component

NeoIDL plugins must comply with a few design rules that `PluginDef` states. `PluginDef` is a Haskell module that basically declares two data types (`Plugin` and `GeneratedFile`) and a type signature (`Transform = Module -> [GeneratedFile]`) defining a family of functions that map a NeoIDL module into a list of files whose contents are the results of the transformation process.

According to these design rules, each NeoIDL plugin must declare an instance of the `Plugin` data type and implement functions according to the `Transform` type signature. Moreover, the `Plugin` instance must be named as `plugin`, so that the `PluginLoader` component will be able to obtain the necessary data for executing a given plugin. Indeed, the execution of a plugin consists of applying the respective transformation function for a NeoIDL module, producing as result a list of files that consists of a name and a `Doc` as file content.⁴

As an alternative, we could have implemented a Haskell type class [10] exposing operations for obtaining the necessary data for a given plugin. Although this approach might seem more natural for specifying design rules for a pluggable architecture in Haskell, in the end it would lead to a cumbersome approach to our problem. The main reason for discarding this alternative approach was the need to (a) implement a data type, (b) make this data type an instance of the mentioned type class, and (c) create an instance of that data type. All those steps would be necessary for each plugin. Using our approach, the obligation of a plugin developer is just to provide an instance of the `Plugin` datatype, taking into account the name convention we mentioned above. The `language` attribute of the `Plugin` datatype is used for UI purpose only, so that the users will be able to obtain the list of available plugins and select which plugins will be used during a program generation.

B. `PluginLoader` component

Based on the design rules discussed in the previous section, the `PluginLoader` component is able to dynamically load the available NeoIDL plugins. This is a Haskell module (see Figure 5) that exposes the `loadPlugins` function, which returns a list with all available plugins. This list is obtained by compiling the Haskell plugin modules during the program execution and dynamically evaluating an expression that yields a list of `Plugin` datatype instances.

We assume that all Haskell modules within the top level `Plugins` directory must have a plugin definition, according to the design rules of Section III-A. In Figure 5, the `loadPlugins` function lists all files within the `Plugins` directory, filters the Haskell files (files with the ``.hs``

```

module PluginLoader (loadPlugins) where
type HSFile = String
dir :: String
dir = "Plugins"
loadPlugins :: IO [Plugin]
loadPlugins =
  let
    pattern = isSuffixOf "hs"
    path file = dir < / > file
    in (list dir) >>= (compile ◦ map path ◦ filter pattern)
dfm = defaultFatalMessenger
flushOut = defaultFlushOut
compile :: [HSFile] → IO [Plugin]
compile modules =
  defaultErrorHandler dfm flushOut $ do
    result ← runGhc (Just libdir) $ do
      let hsModules = map haskellModule modules
          -- five lines of (boilerplate) code are necessary to
          -- dynamically compile Haskell code using GHC
      let exp = buildExpression hsModules
          plugins ← compileExpr (exp ++ " :: [Plugin] ")
          return unsafeCoerce plugins :: [Plugin]
      return result
buildExpression :: [HModule] → String
buildExpression hsms = "[" ++ plugins ++ "]"
where
  plugins = concatMap (λx → x ++ ".plugin") hsms
  concat = join " , "

```

Fig. 5. `PluginLoader` component

extension), creates a qualified name to these files, and applies the `compile` function to the resulting list of qualified names. In the next step, the `compile` function uses the GHC API [12] for compiling the Haskell modules with plugin definitions and to evaluate an expression that produces a list with the available plugins.

Our dynamic approach for loading plugins relies on the GHC API, using a specific idiom to compile Haskell modules and execute expressions. Figure 5 shows that idiom in the definition of the `compile` function, although we omit some boilerplate code that is necessary to compile Haskell modules using the GHC API. The last four lines of `compile` are specific to the program generator of NeoIDL. First, we build a string representation of a Haskell list comprising all instances of the `Plugin` datatype, obtained from the different NeoIDL plugins. Then, we evaluate this string representation of a plugin list using the *meta-programming* ability of the `compileExpr` function, which is available in the GHC API. Thus, `compileExpr` dynamically evaluates a string representation of an expression, which leads to a value that could be used by other functions of a program. The call to `compileExpr` also checks the design rule that requires (a) a plugin definition within all NeoIDL plugins; and (b) that definition must be an instance of the `Plugin` data type. In the cases where a plugin (exposed as a Haskell module on the top-level `Plugins` directory) does not comply with this design rule, a runtime error occurs. Accordingly, we use the default error handler of GHC API to report problems when loading a plugin. This is a new approach of using the GHC API to dynamically check Haskell modules in pluggable architectures.

⁴The `Doc` data type comes from the John Hughes Pretty Printer library.

IV. EVALUATION

In this section we describe an evaluation of the NeoIDL approach through the development and generation of services in the context of a Brazilian Army project. In summary, this evaluation aims at (a) understanding the NeoIDL benefits under the ROI perspective and (b) reasoning about the modular mechanisms of NeoIDL design.

A. The use of NeoIDL in a real context

We have developed nine services that implement operations related to the domain of Command and Control (C2) [1]. These services comprehend almost 50 resources and 3000 lines of Python code. Therefore, all these services have been implemented in Python, though other projects have been implemented in Java as well.

Approximately, the number of lines of Python code related to our service repository increases according to the function $sloc = 330 \times numberOfServices$ — since, in average, each service requires about 330 lines of Python code (with a standard deviation of 119). It is important to note that services are often implemented as a thin layer on top of existing components that implement reusable tasks or business logic. Accordingly, to understand the impact of NeoIDL accurately, here we do not consider lines of code related to (a) existing tasks and business logic implementations and (b) libraries that might be reused through different services.

Based on the development of these services, we estimate that it is possible to generate about 30% to 50% of a service code using NeoIDL. Indeed, in the cases that a service is *data-oriented*, involving basic operations for creating, updating, querying and deleting data, we achieve a higher degree of code generation. Differently, in the cases that a service encapsulates low level behavior (such as the implementation of a *chat-based* message protocol), we achieve a low degree of code generation using NeoIDL, mainly because the current version of NeoIDL does not provide any behavioral construct.

B. Return on Investment of NeoIDL

It is important to reason about the instant in which the design and development of a DSL pays off, since the related effort could not justify the benefits. Accordingly, here we discuss about this issue relating effort to *source lines of code* (SLOC) [15].

NeoIDL comprises almost 2500 lines of code, considering the AST code generated by `BNFConverter`. Note that nearly 67% of the Haskell code results from the `BNFConverter` parser generator. Therefore, excluding the generated code from our analysis, as well as unit testing code and make files, NeoIDL consists of 740 lines of Haskell code and 50 lines of code that (a) specifies the concrete syntax of NeoIDL and (b) serves as input to the `BNFConverter`. According to the COCOMO model [2], it is possible to compute effort from SLOC using equations (1) and (2). This leads to an effort estimation of 3.17 months, which is quite close to the real effort to implement NeoIDL, even considering that a significant effort on the design of NeoIDL was related to the successive refinements on the concrete syntax of the language.

$$\begin{aligned} personMonths &= 2.4 \times KSLOC^{1.05} & (1) \\ &= 2.4 \times 0.79^{1.05} \\ &= 1.87 \end{aligned}$$

$$\begin{aligned} months &= 2.5 \times personMonths^{0.38} & (2) \\ &= 2.5 \times 1.87^{0.38} \\ &= 3.17 \end{aligned}$$

For generating the Python services to the C2 domain, the following NeoIDL modules are necessary: `bnf`, `loader`, `pluginDef`, `main`, `swaggerPlugin`, and `pythonPlugin`. These modules totalize 640 of Haskell and BNF code. Considering the discussion present in the previous section, we estimate the break-even of NeoIDL according to equations (3), (4), and (5). The third equation computes the lines of code necessary for n services without using NeoIDL; whereas the fourth and fifth equations compute the lines of code for n services, considering that NeoIDL generates 30% and 50% of the code, respectively. Therefore, the break-even of NeoIDL must be achieved after developing a number of services between 4 and 7. As a consequence, we believe that the design and development of NeoIDL improve software quality and productivity— by reducing the need to write boilerplate code, at no significant additional costs.

$$sloc = 330 \times numberOfServices \quad (3)$$

$$sloc = 0.7 \times 330 \times numberOfServices + 640 \quad (4)$$

$$sloc = 0.5 \times 330 \times numberOfServices + 640 \quad (5)$$

C. Modularity analysis

NeoIDL includes facilities to develop plugins and to evolve NeoIDL specifications through annotations. As explained in Section III we expose plugins according to some design rules, which allow us to develop and test plugins with a slight dependency on the existing code of the program generator. This encourages contributions to NeoIDL, by enabling developers to design and implement new plugins. In addition, it is possible to unit test a NeoIDL plugin in an isolated manner. Here we relate modularity to extensibility (it is easy to contribute to NeoIDL without a deep knowledge of the core components of NeoIDL) and testability (it is possible to test each NeoIDL plugin in isolation).

Actually, to develop a plugin, it is only necessary to understand the design rules discussed in Section III and an external library (the John Hughes and Simon Peyton Jones Pretty Printer library). For instance, Figure 6 shows the full implementation of a NeoIDL plugin, which reports basic metrics of size from a NeoIDL specification. To keep things simple, that plugin generates a file (named `metrics.data`) whose content consists of the name of a NeoIDL module followed by three lines stating the number of enums, entities, and resources within that module.


```

module Plugins.Metrics (plugin) where
import NeoIDL.Lang.AbsNeoIDL
import PluginDef
import Text.PrettyPrint.HughesPJ
plugin :: Plugin
plugin = Plugin {
  language = "Metrics",
  transformation = generateMetrics
}
print :: String → [a] → Doc
print str lst = text str < + > (text ∘ show ∘ length) lst
generateMetrics :: Transformation
generateMetrics = λ(Module (Ident s) _ _ ens ess rss) →
let
  outputFile = GeneratedFile name content
  name = "metrics.data"
  content = vcat [text "Module" < + > text s
    , print "-enums:" ens
    , print "-entities:" ess
    , print "-resources:" rss]
in [outputFile]

```

Fig. 6. A simple plugin for exporting metrics of a NeoIDL specification

V. RELATED WORK

Many approaches for distributed systems consider the use of an IDL, as discussed in Section I. However, similarly to CORBA [13], WSDL [3], Apache Thrift [17], and Swagger [18], the current version of NeoIDL does not support any construct for specifying formal constraints. Nevertheless, we envision that introducing the semantics of behavioral specification languages (such as Java Modeling Language [11]) into NeoIDL would (a) increase the effectiveness of program generation and (b) enable test case generation from NeoIDL specifications. It is also important to note that two shortcomings of WSDL and Swagger (lack of modularity mechanisms and low expressiveness) motivated the design of NeoIDL, which considered the syntax of other languages (CORBA, Apache Thrift) as inspiration.

Czarnecki and Eisenecker present many approaches for Generative Programming [4], including Aspect-Oriented Programming, C++ Template Metaprogramming, and Domain Specific Languages. NeoIDL comprises a domain specific language for services' description and a pluggable architecture with an extension point that allows code generation for different target languages. Although several works describe the use of Haskell to implement (embedded) domain specific languages [9], the use of Haskell to build pluggable architectures has not been extensively discussed in the literature. Similar to the `hs-plugins` framework [14], NeoIDL architecture uses the infrastructure of the Glasgow Haskell Compiler to dynamically load and compile Haskell modules that implement NeoIDL plugins.

VI. FINAL REMARKS AND FUTURE WORK

This paper introduced NeoIDL, a domain specific language for service specifications. We discussed the design and implementation of NeoIDL, which comprises a specification language and a pluggable architecture for generating code for

different languages. We further discussed the main contributions of NeoIDL with respect to existing interface description languages (such as CORBA IDL and WSDL)—NeoIDL provides means for language extensibility and specification modularity. As a future work, we aim at writing NeoIDL plugins to generate code to other web frameworks, such as Play and Yesod Frameworks. We also intend to investigate the use of behavioral specification constructs in NeoIDL, so that we could generate test cases from NeoIDL specifications.

ACKNOWLEDGMENT

This work was partially supported by a research collaboration project between the Brazilian Army and the University of Brasilia (project name: GEPRO EXERCITO TDC EVOLUCAO CORTEX 2012).

REFERENCES

- [1] Alberts, D.S., Hayes, R.E.: Understanding Command and Control. DoD Command and Control Research Program, 1st edn. (2006)
- [2] Boehm, B.W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., Steece, B.: Software Cost Estimation with Cocomo II. Prentice Hall PTR, 1st edn. (2000)
- [3] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (wsdl) 1.1. W3C recommendation, W3C (Feb 2001), <http://www.w3.org/TR/wsdl>
- [4] Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (2000)
- [5] Erl, T., Balasubramanian, R., Carlyle, B., Pautasso, C.: SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST. Prentice Hall (2012)
- [6] Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
- [7] Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Technol. 2(2), 115–150 (May 2002)
- [8] Hadley, M.: Web application description language (wadl). W3C recommendation, W3C (Aug 2009), <http://www.w3.org/Submission/wadl/>
- [9] Hudak, P.: Building domain-specific embedded languages. ACM Computing Surveys (CSUR) 28(4es), 196 (1996)
- [10] Jones, M.P.: Functional programming with overloading and higher-order polymorphism. In: Jeuring, J., Meijer, E. (eds.) Advanced Functional Programming, Lecture Notes in Computer Science, vol. 925, pp. 97–136. Springer (1995)
- [11] Leavens, G.T., Baker, A.L., Ruby, C.: Preliminary design of jml: A behavioral interface specification language for java. Softw. Eng. Notes 31(3), 1–38 (May 2006)
- [12] Marlow, S., Peyton-Jones, S.: The Glasgow Haskell Compiler. In: Brown, A., Wilson, G. (eds.) The Architecture of Open Source Applications, vol. 2. lulu.com (2012)
- [13] (OMG), O.M.G.: Interface definition language 3.5. Tech. rep., Object Management Group (2014), <http://www.omg.org/spec/IDL35/3.5/PDF/>
- [14] Pang, A., Stewart, D., Seefried, S., Chakravarty, M.M.T.: Plugging haskell in. In: Proceedings of the 2004 ACM SIGPLAN Workshop on Haskell. pp. 10–21. Haskell '04, ACM, New York, NY, USA (2004)
- [15] Park, R.: Software size measurement: A framework for counting source statements. Tech. Rep. CMU/SEI-92-TR-020 (1992)
- [16] Ranta, A.: Implementing Programming Languages. An Introduction to Compilers and Interpreters. Texts in computing, College Publications (2012)
- [17] Slee, M., Agarwal, A., Kwiatkowski, M.: Thrift: Scalable cross-language services implementation. Tech. rep., Facebook (2012), <http://thrift.apache.org/static/files/thrift-20070401.pdf>
- [18] Team, S.: Swagger restful api documentation specification 1.2. Tech. rep., Wordnik (2014), <https://github.com/wordnik/swagger-spec/blob/master/versions/1.2.md>

A Unified MapReduce Domain-Specific Language for Distributed and Shared Memory Architectures

Daniel Adornes, Dalvan Griebler, Cleverson Ledur, Luiz Gustavo Fernandes
Pontifical Catholic University of Rio Grande do Sul (PUCRS),
Faculty of Informatics (FACIN), Computer Science Graduate Program (PPGCC),
Parallel Application Modeling Group (GMAP).

Av. Ipiranga, 6681 - Building 32 - Porto Alegre - Brazil

{daniel.adornes,dalvan.griebler,cleverson.ledur}@acad.pucrs.br, luiz.fernandes@pucrs.br

Abstract—MapReduce is a suitable and efficient parallel programming pattern for processing big data analysis. In recent years, many frameworks/languages have implemented this pattern to achieve high performance in data mining applications, particularly for distributed memory architectures (e.g., clusters). Nevertheless, the industry of processors is now able to offer powerful processing on single machines (e.g., multi-core). Thus, these applications may address the parallelism in another architectural level. The target problems of this paper are code reuse and programming effort reduction since current solutions do not provide a single interface to deal with these two architectural levels. Therefore, we propose a unified domain-specific language in conjunction with transformation rules for code generation for Hadoop and Phoenix++. We selected these frameworks as state-of-the-art MapReduce implementations for distributed and shared memory architectures, respectively. Our solution achieves a programming effort reduction from 41.84% and up to 95.43% without significant performance losses (below the threshold of 3%) compared to Hadoop and Phoenix++.

Keywords: *MapReduce, Domain-Specific Language, Parallel Programming, Effort Evaluation, Performance Evaluation.*

I. INTRODUCTION

An exponential volume of data is generated by a variety of fields worldwide, for example, social networks, governments, health care, stock market, among others. The so-called *Big Data* is addressed by data analysis applications, which may imply high computational costs. Consequently, high-performance computing is needed to process all data in time. Google initially proposed a solution for improving the performance of these application's domain, by combining *Map* and *Reduce* operations as a single parallel pattern named MapReduce [5]. Since then, the MapReduce has originated many implementations by both industry and academic research. Some of them have achieved great importance, such as Hadoop¹, which is suited for programming in large clusters architectures, and Phoenix++ [13] for programming in multi-core architectures.

MapReduce is a high-level pattern concept for expressing parallelism and taking advantage of different parallel architectures [5]. However, current state-of-the-art implementations

impose additional complexities beyond this pattern, requiring developers to deal with low-level programming aspects, such as memory management and network communication. Moreover, there are host language prescriptions imposed by the programming interface of library-based approaches. These aspects motivate a particular language syntax for MapReduce implementation.

This paper proposes a unified domain-specific language to reduce the programming effort and improve code reuse between distributed and shared memory architectures. Code transformation rules are also proposed together with a transformation process aimed at being fully compliant with the key features of original MapReduce solutions.

The contributions are the following:

- A unified MapReduce domain-specific language for parallel and distributed architectures.
- A programming interface approach that significantly reduces the development effort.
- An efficient set of code transformation rules for Hadoop and Phoenix++ without significant performance loss.

The paper is organized as follows. Section II discusses the most important related work. Section III details the proposed domain-specific language. Section IV describes the methodology approached for the evaluation. Section V performs the experiments and evaluates the performance and programming effort results. Finally, Section VI presents the conclusions and future works.

II. RELATED WORK

We aim at providing a unified programming interface to reduce programming effort and allow code reuse between distributed and shared memory architectures. A Domain-Specific Language (DSL) approach allows programmers to focus on specific domains [6]. In this paper, we propose an external DSL consisting of an entirely new language. Related work, in turn, are embedded DSLs, which restricts their programming interface's flexibility and abstraction.

Hadoop was the first widely used MapReduce implementation, aimed at processing large data sets in distributed systems. It provides a Java API for defining *Map* and *Reduce* logic, which are run by distributed computation components over distributed storage components. The distributed storage

¹<http://hadoop.apache.org>

DOI reference number: 10.18293/SEKE2015-204

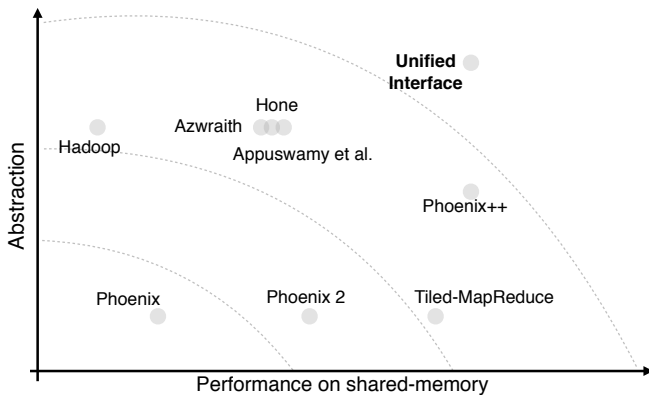


Fig. 1: The relationship graph between abstraction and performance on the programming interface design goals.

components rely on Hadoop Distributed File System (HDFS), which provides the vision of a single file system for large data sets stored on all nodes in the cluster.

Ranger et al. [11] proposed the first version of Phoenix as an optimized implementation of MapReduce for shared-memory architectures, with a C-based programming interface. Yoo et al. [16] evolved the Phoenix project with new optimizations for multi-core architectures with non-uniform memory access. Chen et al. [12], [4] proposed a new implementation of Phoenix, based on a tiled (iterative) approach, named Tiled-MapReduce. Finally, Talbot et al. [13] proposed Phoenix++, consisting of a completely rewritten version of Phoenix, implemented in C++ and taking advantage of the language’s capabilities for modularity and code reuse in order to allow a more adaptive programming interface.

Phoenix++ was mainly motivated by the recurrent need of customizations reported by many of its users while working with different applications. Talbot et al. realized that object-oriented capabilities of C++ could be exploited for creating specialized *containers* for storage of intermediate data and *stateful combinators* for storing the cumulative value of associative *reduction* operations (e.g., sum, product).

Concerning to high-performance, Phoenix++ outperforms all its predecessors, previously mentioned. Also, compared to Hadoop, Phoenix++ achieves a 28.5x speedup while executing on a single machine (not distributed). It motivated us to use Hadoop for distributed memory and Phoenix++ for shared-memory multi-core architectures.

Recently, some researches [15], [2], [10] worked on improving Hadoop’s performance on the single-node level, mainly by avoiding some internal mechanisms not needed for non-distributed environments (e.g., message passing and replication) and harnessing the computational power of multi-core. These researches also kept the Hadoop’s programming interface, thus not adding a new abstraction layer. Their evaluations, though, show that Phoenix++ still outperforms in the single-node level.

Some important implementations addressing heterogeneous architectures, with CPU and GPU, were also analyzed but are

still not covered by our transformation process. From these, Grex [3] is the state-of-the-art, providing a specific API for controlling the MapReduce phases and the way data structures are stored in the different memory levels of GPUs.

Through a relationship graph, Figure 1 demonstrates our understanding of the related work achievements concerning abstraction and performance. It is possible to visualize that most programming interfaces designed for distributed architectures provide higher abstraction compared to those for multi-core. Such difference is related to programming aspects and their goals. In shared memory, programmers are required to deal with low-level mechanisms such as pointers and memory allocation in order to maximize the usage of very restricted resources. By other hand, resources are much less restricted in distributed architectures, allowing Hadoop’s interface to be favored by using Java as host language.

Moreover, Figure 1 also justifies our choices for generating MapReduce code. Hadoop and Phoenix++ are the best alternatives for our design principles in terms of abstraction and performance. The following section discusses these and other design aspects for the proposed solution.

III. THE PROPOSED DOMAIN-SPECIFIC LANGUAGE

A unified MapReduce programming interface is proposed in conjunction with code transformation rules for Phoenix++ and Hadoop MapReduce.

The proposed interface is inspired on the building block syntax proposed by Griebler et al. [8], [7], since it demonstrated significant effort reduction for the Master/Slave parallel pattern. Our interface however is not built over a third-part language as C or C++, being based on an own language instead. The different programming languages (Java and C++) and the very specific syntaxes for Phoenix++ and Hadoop code led us to decide for an own language in order to maximize abstraction. A C++ programming interface provided by the Hadoop project, called Hadoop Pipes², was initially considered but later discarded due to absence of documentation and for looking as a discontinued project.

The interface’s structure consists of an outer `@MapReduce` block and two inner `@Map` and `@Reduce` blocks, as detailed on listing 1 and grammar 1.

```

1 @MapReduce<NAME, K_IN, V_IN, K_OUT, V_OUT, K_DIST>{
2   @Map(key, value){
3     // Map code logic
4   }
5   @Reduce(key, values){
6     // Reduce code logic
7   }
8 }

```

Listing 1: Interface’s structure.

The `@MapReduce` block always requires six parameters, namely `NAME`, `K_IN`, `V_IN`, `K_OUT`, `V_OUT` and `K_DIST`.

The `NAME` parameter is any user-defined name, which is used for identifying the MapReduce process and further transforming the code for Java and C++ classes.

²<http://wiki.apache.org/hadoop/C++WordCount>

```

<Map> ::= '@Map' 'C' <key> , <value> ')' '{' { <cmd>* <EmitCall> <cmd>* }
'}'
<Reduce> ::= '@Reduce' 'C' <key> , <values> ')' '{' { <cmd>* <EmitCall>
<cmd>* } '}' | '@SumReducer' | '@IdentityReducer'
<MapReduce> ::= '@MapReduce' '<' <mapreduce-params> '>' '{' <Map>
<Reduce> '}'

```

Grammar 1: Structure's grammar

The *K_IN*, *V_IN*, *K_OUT* and *V_OUT* parameters are used to define the *<key/value>* input and output types, respectively. In other words, these parameters define which type of raw data is initially read by the MapReduce process and which type of reduced data is produced by it at the end.

The *K_DIST* parameter, in turn, is used for defining the keys distribution, whether **,**, **:k* or *1:1*. It is used by Phoenix++ [13] for employing memory-optimized data structure for intermediate *<key/value>* pairs, taking advantage of applications in which the number of keys to be emitted is known in advance.

The inner blocks, *@Map* and *@Reduce*, must be programmed by the user in order to define the core logic of the given MapReduce application. The *@Map* block receives a *<key/value>* input pair from which to compute the *<key/value>* intermediate pairs. Finally, the *@Reduce* block receives all mapped values for each key, this is a *<key/values>* pair, and computes the final reduced *<key/value>* pair by key. Both blocks are provided with an *emit* function (grammar 2), which for *@Map* block represents the function to emit intermediate *<key/value>* pairs and for *@Reduce* block represents the function to emit the final reduced value for a given key.

```

<EmitCall> ::= 'emit' 'C' <key> , <value> ')'

```

Grammar 2: The emit function's grammar

One additional characteristic of the *@Reduce* block is that it can be replaced by a single *@SumReducer* directive with no block code (grammar 1), which indicates that a simple sum operation must be performed over all values of each key. Another option is the *@IdentityReducer* directive, which indicates that no reduction needs to be performed. Both default options are also provided by Hadoop and Phoenix++, since these are common reduce logics for MapReduce applications. Nevertheless, whenever the provided default reducers do not fit the need, a customized reducer can be implemented, as demonstrated in listing 2 for a sample multiplicand reducer.

```

1 @Reduce(key, values){
2   double product = 1
3   for(int i=0; i < length(values); i++)
4     product *= values[i]
5   emit(key, product)
6 }

```

Listing 2: Multiplicand reducer with proposed interface.

Finally, listing 3 demonstrates the code of a Histogram application with the proposed interface. This sample implementation uses the *@Type* directive to define a variable

type *pixel* that stores the RGB values for each pixel in the processed image. Then, the *@MapReduce* directive defines the name Histogram and the four variable types for *<key, value>* input and output pairs. The **:768* indicates that it is known in advance that a maximum of 768 distinct keys will be emitted by the *Map* phase. This information optimizes the data structure used by the generated Phoenix++ code to hold the intermediate *<key, value>* pairs. Finally, a *@Map* block defines that *map* phase will emit value 1 for each occurrence of a given color in the RGB of the pixel being processed, and the *@SumReducer* defines that reduction will be performed as a simple sum operation over all emitted values for each distinct key.

```

1 @Type pixel(r: ushort, g: ushort, b: ushort)
2 @MapReduce<Histogram, long, pixel, int, ulonglong,
3   " *:768 ">{
4   @Map(key, p){
5     emit(p.b, 1)
6     emit(p.g+256, 1)
7     emit(p.r+512, 1)
8   }
9   @SumReducer
10 }

```

Listing 3: Histogram with proposed interface.

A. Interface components and transformation rules

For developing the *@Map* and *@Reduce* logics the programmer is provided with a set of proposed interface's components, which comprehends variable types, built-in functions and flow control structures. Each of these components has an associated transformation rule, through which its equivalent component in Hadoop and Phoenix++ can be later generated.

Variable types can also be custom types defined by the programmer with the *@Type* keyword, which are translated to C++ *structs* for Phoenix++ and Java classes implementing the *WritableComparable* interface for Hadoop. The resulting Java classes, particularly, include *getters* and *setters* methods, besides some other methods whose implementation is required by *WritableComparable* interface.

Moreover, whenever a custom type is defined for input data (*V_IN*) in Hadoop, a complete implementation of a subclass of *FileInputFormat* and another subclass of *RecordReader* is required. It is particularly needed in order to instruct Hadoop on how to split and distribute the input data among *Map* tasks. Nonetheless, it causes applications developed with Hadoop to reach a considerable amount of code. The generated subclass of *RecordReader* considers each line of an input file to represent a single instance of the given custom type.

Also, whenever a custom type is used for output values (*V_OUT*), Phoenix++ requires the implementation of a custom *associative_combiner*, which in turn is most likely to perform a simple sum for internal attributes of the custom type. By assuming this, the unified interface still allows *@SumReducer* directive even if output values are of a custom type. In this case, the code transformation is defined for the correspondent *associative_combiner* and *Reducer* class of Phoenix++ and Hadoop respectively.

Finally, Phoenix++ requires specific types (*structs*) and a complex *split* logic for text processing applications. In the proposed interface, whenever the type *Text* is chosen as input value (*V_IN*), the transformation rules automatically include such components in the C++ generated code.

B. Transformation Process

Transformation rules are applied through a transformation process, whose stages are described as follows:

- **First stage** - The process starts by generating *imports* (for Hadoop Java code) and *includes* (for Phoenix++ C++ code) always required by any application. It consists of base libraries of these frameworks.
- **Second stage** - The process then continues by transforming the *@MapReduce* block and its *@Map* and *@Reduce* inner blocks. At this same stage, custom types *@Type* may have been provided by the programmer being then also transformed. Ultimately, global variables, external to the *@MapReduce* block, may have also been defined by the programmer and are also transformed in this second stage.
- **Third stage** - This stage addresses transformations derived from the input and output keys and values, interpreted in the second stage (e.g., text processing components previously mentioned).
- **Fourth stage** - This stage transforms the variable types defined in the blocks' signature and also internally to these blocks.
- **Fifth stage** - Ultimately, the fifth stage transforms the functions defined externally to the MapReduce blocks or internally to custom types.

The proposed process is based on Aho et al. [1], thus consisting of language recognition, analysis and code generation. Language recognition phase comprehends the interpretation through lexical, syntactic and semantic analysis. Lexical analysis validates the compliance with the proposed components then producing tokens. The syntactic analysis uses the identified tokens to check the grammar language and report syntax errors. The semantic analysis in turn checks how components are disposed throughout the whole code. An overview of the transformation flow is shown in figure 2. Effective transformations and code generation are proposed as future work (section VI).

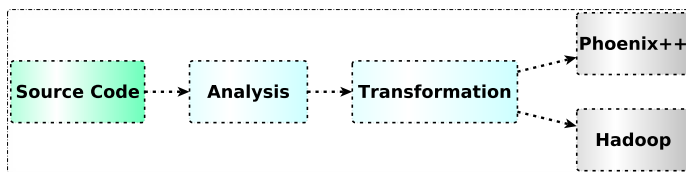


Fig. 2: Domain-Specific Language Flow.

Along the language recognition phase, an AST (Abstract Syntax Tree) is created. An AST is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construction present in the source code. AST creation is

bottom-up because the nodes addition starts from the smaller tokens and patterns. Finally, AST stores the identified tokens for later use in the code generation phase, which then traverses the tokens in the AST in order to generate new code in the target language.

IV. METHODOLOGY

We evaluated our DSL using two approaches. First we evaluated its interface using SLOccount³ to measure programming effort. The second approach consisted of performance results. Five different applications with specific peculiarities, namely Word Count, Word Length, Histogram, K-means and Linear Regression, were implemented with the proposed interface and generated through the transformation rules for Phoenix++ and Hadoop. This applications were also implemented purely in Phoenix++ and Hadoop.

The main peculiarity we looked for while choosing the sample applications was the key distribution. Word Count demonstrates the **.** distribution, whereas Word Length, Histogram, K-means and Linear Regression demonstrate the **:k* distribution. Additionally, other peculiarities are also covered by the selected sample applications, such as custom types and custom combiners.

The Matrix Multiplication and PCA (Principal Component Analysis applications) would fit the *1:1* distribution, however would also require more programming controls for Phoenix++ generated code beyond the abstraction aimed by the proposed interface and with no equivalent functionality in Hadoop.

A. Effort Evaluation

For programming effort measurement it was used the SLOccount suite, also used by Griebler et al. [7] for the evaluation of DSL-POPP and by a set of other researches (e.g., [9], [14]). SLOccount³ is a software measurement tool, which counts the physical source lines of code (SLOC), ignoring empty lines and comments. It also estimates development time, cost and effort based on the original Basic COCOMO⁴ model.

The suite supports a wide range of both old and modern programming languages (e.g., C++ and Java), which are naturally inferred by SLOccount and thus used for measurement. For our unified interface, we selected C++ because it has similar syntax.

B. Performance Evaluation

For evaluating performance, the workload for Word Count and Word Length was a 2Gb text file, for Histogram, a 1.41 Gb image with 468,750,000 pixels and for Linear Regression the workload was a 500Mb file. For Kmeans, no input file is required, since number of points, means, clusters and dimensions are parametrized through command line or assumed to the default values of 100,000, 100, 3 and 1,000, respectively, which were considered for our tests. All workloads are available at the Phoenix++ project's on-line repository⁵.

³<http://www.dwheeler.com/sloccount/sloccount.html>

⁴<http://www.dwheeler.com/sloccount/sloccount.html#cocomo>

⁵<https://github.com/kozyraki/phoenix>

For performance evaluation of generated Phoenix++, we used a multi-core system with a 2.3 GHz Intel Core i7 processor, four cores with Hyper-Threading and 16Gb of DRAM, whereas for performance evaluation of generated Hadoop, we used 8 nodes of a cluster, where each node is equipped with a 2.4 GHz Intel Xeon Six-Core E5645 processor and 24Gb of DRAM. The cluster sums 192 cores, where the nodes are interconnected by 2 Gigabit-Ethernet networks and 2 InfiniBand networks.

In order to obtain the arithmetic means, 30 execution times were collected for each sample application such as described in Tables II and I, and Figures 4 and 3. For running Hadoop applications, we used a synthetic script ⁶ and copied to HDFS all data so that all cluster nodes had access.

V. RESULTS

As described in section IV, we evaluated the proposed DSL using two approaches for measuring programming effort and performance. Tables I and II show the mean execution time for each sample application for the code transformed from our proposed unified interface and for the code developed directly from Phoenix++ and Hadoop, respectively. Figures 3 and 4 graphically demonstrate these same measurements.

TABLE I: Original and generated Phoenix++.

	WC	WL	Histogram	Kmeans	LR
Original	5.38	4.02	2.83	5.98	0.62
Generated	5.37	3.99	2.87	6.09	0.63
Difference	-0.27%	-0.9%	1.4%	1.7%	0.3%

TABLE II: Original and generated Hadoop.

	WC	WL	Histogram	Kmeans	LR
Original	36.24	26.36	21.87	51.36	5.97
Generated	37.22	26.48	22.42	50.59	6.01
Difference	2.63%	0.45%	2.45%	-1.52%	0.76%

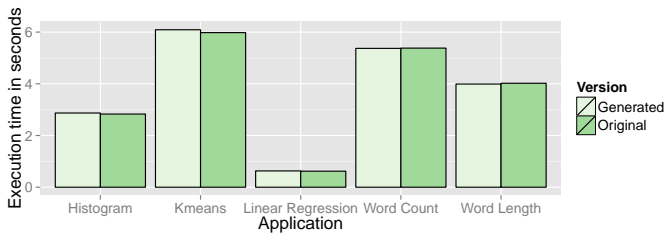


Fig. 3: Original and generated Phoenix++.

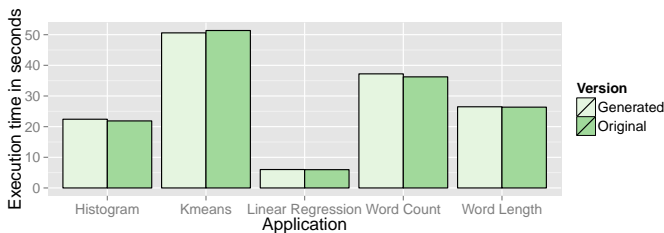


Fig. 4: Original and generated Hadoop.

⁶<https://github.com/mvneves/hadoop-deploy>

Through tables I and II and figures 3 and 4 it is possible to visualize the negligible difference (less than 3%) between the execution time of the generated and original versions for the two frameworks. Performance losses are considerably avoided as a direct result of the effective coverage of performance components by the transformation rules.

Tables III and IV and figures 5 and 6 show the SLOC and cost measurements. It is possible to observe that the difference between the measurements of SLOC and Cost is negligible, which confirms the approach used by COCOMO model.

A significant SLOC reduction can be observed for *Word Count* and *Word Length* applications compared to Phoenix++ code, which take advantage of the specific components for text processing applications. For such applications, Phoenix++ requires a wide set of mechanisms whose need is then identified in advance by the proposed transformation rules, being it transparent while developing with the proposed unified interface.

TABLE III: SLOC count reduction

Application	Phoenix++	Hadoop	Unified Interface	Reduction compared to Phoenix++	Reduction compared to Hadoop
WordCount	89	27	8	91.01%	70.37%
WordLength	95	33	14	85.26%	57.58%
Histogram	22	170	9	59.09%	94.71%
K-means	98	244	57	41.84%	76.64%
Linear Regression	31	171	18	41.94%	89.47%
	67	129	21.2	63.83%	77.75%

TABLE IV: Cost estimate reduction

Application	Phoenix++	Hadoop	Unified Interface	Reduction compared to Phoenix++	Reduction compared to Hadoop
WordCount	\$2,131.00	\$609.00	\$170.00	92.02%	72.09%
WordLength	\$2,282.00	\$752.00	\$306.00	86.59%	59.31%
Histogram	\$491	\$4,204.00	\$192.00	60.90%	95.43%
K-means	\$2,357.00	\$6,143.00	\$1,334.00	43.40%	78.28%
Linear Regression	\$704.00	\$4,229.00	\$398.00	43.47%	90.59%
	\$ 1,593.00	\$ 3,187.20	\$ 480.00	65.28%	79.14%

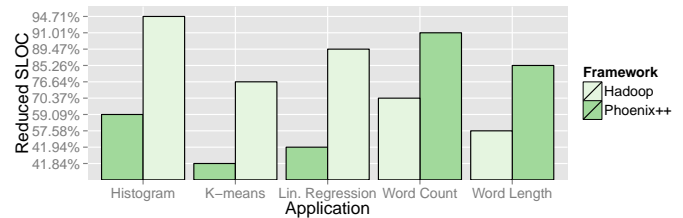


Fig. 5: SLOC count reduction

Compared to Hadoop, the *Histogram*, *K-means* and *Linear Regression* applications achieved greater SLOC reduction mainly for the amount of code required to treat custom types (subclasses of *WritableComparable*) in Hadoop.

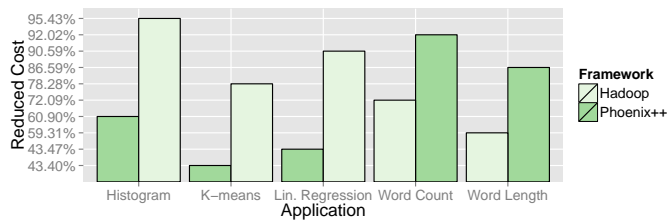


Fig. 6: Cost estimate reduction

K-means also takes advantage over Phoenix++ by completely avoiding code for custom combiner, however many functions are required by the sample implementation, which causes little SLOC reduction with the unified interface.

VI. CONCLUSIONS

From selecting Phoenix++ and Hadoop as the state-of-the-art solutions for shared-memory and distributed architectures, respectively, this work proposes a solution for abstracting MapReduce programming without losing the performance optimizations of these selected implementations. Such objective is achieved through a unified MapReduce programming interface, proposed in conjunction with a comprehensive set of transformation rules for Phoenix++ and Hadoop.

Except for a specific data locality configuration for NUMA systems provided by Phoenix++, the transformation rules are effective in covering from custom types to custom functions, custom combiners, default reducers, different key distributions and text processing components, covering thus all components needed from the selected sample applications. Moreover, performance losses are successfully avoided (difference of less than 3%) and SLOC and cost reduction indicates that programmers' productivity can be considerably increased.

Some advantages and main contributions are the reuse of code between different architectures and the possibility of expanding the coverage of the transformation rules to other MapReduce solutions and architectural levels.

A limitation is that programmers are still required to implement the code to call the MapReduce process, thus being required to know C++ and/or Java. However, some on-line services, such as Amazon's Elastic MapReduce⁷ (EMR), require only the Hadoop MapReduce implementation, abstracting the invocation code from developers. Nonetheless, we conclude that the SLOC and cost reduction achieved by the proposed interface compensate such limitation.

As future work we plan to expand the transformation rules in order to cover MapReduce solutions such as Grex [3] for heterogeneous parallel architectures. Finally, we also visualize an expansion of the DSL's programming interface, particularly by adding more built-in functions and variable types.

VII. ACKNOWLEDGMENTS

This work was supported by FAPERGS (Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul), CAPES

(Coordenação de Aperfeiçoamento Pessoal de Nível Superior), FACIN (Faculdade de Informática) and PPGCC (Programa de Pós-Graduação em Ciência da Computação).

REFERENCES

- [1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.
- [2] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. Scale-up vs Scale-out for Hadoop: Time to Rethink? In *Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC '13*, pages 20:1–20:13, Santa Clara, CA, October 2013. ACM.
- [3] C. Basaran and K.-D. Kang. Grex: An efficient MapReduce Framework for Graphics Processing Units. *J. Parallel Distrib. Comput.*, 73(4):522–533, May 2013.
- [4] R. Chen and H. Chen. Tiled-MapReduce: Efficient and Flexible MapReduce Processing on Multicore with Tiling. *ACM Trans. Archit. Code Optim.*, 10(1):3:1–3:30, April 2013.
- [5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, Berkeley, CA, USA, December 2004. USENIX Association.
- [6] M. Fowler. *Domain-Specific Languages*. Addison-Wesley, Boston, USA, 2010.
- [7] D. Griebler, D. Adornes, and L. G. Fernandes. Performance and Usability Evaluation of a Pattern-Oriented Parallel Programming Interface for Multi-Core Architectures. In *The 26th International Conference on Software Engineering & Knowledge Engineering*, pages 25–30, Vancouver, Canada, July 2014. Knowledge Systems Institute Graduate School.
- [8] D. Griebler and L. G. Fernandes. Towards a Domain-Specific Language for Patterns-Oriented Parallel Programming. In *Programming Languages - 17th Brazilian Symposium - SBLP*, volume 8129 of *Lecture Notes in Computer Science*, pages 105–119, Brasilia, Brazil, October 2013. Springer Berlin Heidelberg.
- [9] M. Hertz, Y. Feng, and E. D. Berger. Garbage Collection Without Paging. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '05*, pages 143–153, New York, NY, USA, 2005. ACM.
- [10] K. A. Kumar, J. Gluck, A. Deshpande, and J. Lin. Optimization Techniques for "Scaling Down" Hadoop on Multi-Core, Shared-Memory Systems. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT '14*, pages 13–24, Athens, Greece, 2014. Open Proceedings.
- [11] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, HPCA '07*, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.
- [12] H. C. Rong Chen and B. Zang. Tiled-MapReduce: Optimizing Resource Usages of Data-Parallel Applications on Multicore with Tiling. In *Proc. of the 19th Int'l Conference on Parallel Architectures and Compilation Techniques*, page 523–534, Vienna, Austria, September 2010.
- [13] J. Talbot, R. M. Yoo, and C. Kozyrakis. Phoenix++: Modular MapReduce for Shared-Memory Systems. In *Proceedings of the second international workshop on MapReduce and its applications*. MapReduce '11, pages 9–16, San Jose, California, USA, May 2011. ACM.
- [14] N. Vazou, E. L. Seidel, R. Jhala, D. Vytiniotis, and S. Peyton-Jones. Refinement Types for Haskell. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming, ICFP '14*, pages 269–282, New York, NY, USA, August 2014. ACM.
- [15] Z. Xiao, H. Chen, and B. Zang. A Hierarchical Approach to Maximizing MapReduce Efficiency. In *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques, PACT '11*, pages 167–168, Washington, DC, USA, October 2011. IEEE Computer Society.
- [16] R. M. Yoo, A. Romano, and C. Kozyrakis. Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC), IISWC '09*, pages 198–207, Washington, DC, USA, October 2009. IEEE Computer Society.

⁷<http://aws.amazon.com/elasticmapreduce>

Towards a Metamodel Design Methodology

Experiences from a model transformation metamodel design

Magalhães, A.P.; Maciel, R.S.P.; Andrade, A.

Science Computer Department

Federal University of Bahia

Salvador, Brazil

anapatriciamgalhaes@gmail.com

{ritasuzana, aline}@dcc.ufba.br

Abstract— Software engineering makes extensive use of models to provide a better understanding of artifacts produced during system development. Models are specified in modeling languages such as UML or using Domain Specific Languages. In this paradigm of development, metamodeling is essential because it is usually used to specify the abstract syntax of these languages. However, the design of metamodels is not a trivial task, it requires expertise in specific domains, language definition and abstraction capabilities. This paper provides a guide for metamodel design towards a metamodel development methodology based on some lessons learned from metamodel design experiences.

Keywords- *metamodel guide; metamodeling design; metamodel methodology*

I. INTRODUCTION

In software engineering models have been extensively used to provide a better understanding of the artifacts used in system development. A model can be seen as a set of elements that describes a system in a specific purpose [1]. Models are specified conform to modeling languages such as UML or Domain Specific Languages (DSL) [17] and usually the abstract syntax of modeling languages are specified as metamodels. The design of a metamodel requires expertise in metamodeling techniques and knowledge in the domain of the language under construction as well as a good capacity of abstraction [3].

Our research group had been working on many projects that require the definition of metamodels [5][6][7]. In all of these projects we have felt the need for a method to guide us in some issues such as: how to define a metamodel concepts, how to guarantee that a metamodel covers all the desired concepts of the target domain; how to structurally organize the concepts; and how to validate a metamodel.

Some work has been done in Domain Specific Language creation [14][18], about strategies to specify structural aspects of a metamodel [4], and metamodels pattern identification [3][20]. However, most of them do not focus on aspects such as concepts identification and metamodel validation. These aspects are important to guarantee the coverage level of the metamodel when instantiating models. Furthermore, the existing works do not guide developers through the entire development of the metamodel.

This paper presents a proposal to guide developers in metamodel design based on our experiences in developing metamodels. This guide puts together the tasks that our group performed during the development of some metamodels (e.g. how we selected metamodels concepts) and the lessons learned. As these tasks started to be performed in a systemically manner we organized them, step by step, towards a design metamodel methodology. We aim to systematize the tasks involved in metamodel development leveraging the quality of the produced metamodels in terms of coverage of the concept definition, metamodel detailing (e.g. definitions of concept attributes) and organization of these concepts (e.g. use of specializations).

As we have recently designed a metamodel for transformation domain [7], called MMT (MetaModel for Transformation), we used this to explain the proposed guide.

The rest of this paper is organized as follows: section 2 presents the related works; section 3 presents the proposed guide using the design of a transformation metamodel as an example; section 4 presents the validation of the proposed guide; and section 5 presents our conclusions.

II. RELATED WORKS

Nowadays, there are several approaches to help in metamodel design. These approaches can be divided into structural approaches and validation approaches.

In [4] the author gives guidelines for designing metamodels focusing on structural modeling aspects. These guidelines comprise rules to better organize the domain concepts (e.g. how to specialize concepts with similar attributes and associations). In the same vein [3] [20] propose design patterns for metamodels. The authors analyze many different metamodels and identify recurrent problems in the metamodel structures, for example different concepts with the same attributes or relationships. Patterns are suggested to solve these problems e.g. the use of concept generalizations or specializations. When developing a metamodel, developers may use patterns to structure the domain concepts.

In [15] the authors propose a methodology for developing metamodels focusing on simulation based on mathematical statistics techniques. Therefore, this work has a different field of study than ours whose principal objective is the definition of metamodel constructors.

The work proposed in [26] uses Test-Driven Development (TDD) to define and validate metamodels. It represents the requirements of a metamodel as models and uses these models as test cases to perform validations. From the outcome of these validations it incrementally defines the metamodel. Differently, we capture metamodel concepts through examples of models in the referred domain and from comparison of metamodel concepts to existing theories (e.g. taxonomy). Our validation assesses metamodel expressiveness through instantiation of models.

In [27] the authors use elements of generic programming to give solutions for the specification of metamodels concerned to reuse and modularization (e.g. it uses *templates* to define patterns and libraries). In a different direction, our work focuses on the definition of a guide to develop metamodels based on traditional software development life cycle.

There are works focusing on the creation of Domain Specific Language. The book [14] lists many definitions of language, grammar, syntax and semantics, how to implement a parser, what a semantic model is and other aspects related to language creation. Similarly, [18] proposes guidelines for DSLs creation related to concrete syntax (e.g. language representation using textual or graphic notation, redundancies control, and so on). In [16] the authors criticize the use of languages such as MOF on metamodel creation due to the time consumed on development and propose a DSL to design metamodels; and [19] proposes the systematic use of examples to increase quality in domain knowledge definition.

Therefore, these works usually focus on specific aspects of metamodel design and do not provide an integrated solution that covers the definition of metamodel concepts, structural design and validation. Besides this, none of these works provide a guide for developers on metamodel design tasks. Our work aims to cover the development of metamodels from concepts definition to validation. Furthermore, some of these works can be integrated to our proposal as part of some tasks (e.g. we used the guidelines proposed by [4] to better organize the metamodel structural aspects).

III. METAMODEL DESIGN GUIDE

In the absence of a methodology that focuses on metamodel design we began to define metamodels in our laboratory in an ad hoc way generating releases incrementally. However, after some development iterations we observed that the tasks performed during the metamodel definition were almost the same. As a result, we started executing them systematically. We organized these tasks as a guide (specified using SPEM 2.0 metamodel) to help in metamodel development. An overview of this guide is shown in Fig. 1, it comprises three phases: (i) *Conceptual Modeling*; (ii) *Design*; and (iii) *Validation*. Each phase can be executed in many cycles of iteration performing a set of tasks.

Fig. 2 shows a work flow with the tasks of the *Conceptual Modeling* phase: initially the *Domain Knowledge* and *Concepts Identification* tasks are executed to select the relevant concepts in order to initiate the metamodel definition (*Create Metamodel* task). Then, this metamodel can be compared to an existing theory (*Theory Comparison* task) and might be reviewed many

times (*Metamodel Review* task) until the definition of its first release. When necessary it is also possible to return to *Domain Knowledge* task to get some more examples.

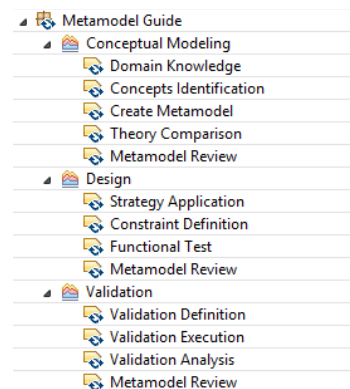


Figure 1. Phases and tasks of the Metamodel development guide.

According to SPEM, a task can be performed in a set of steps and may consume / produce work products. Besides this, roles are responsible for the tasks. For each task of the guide we specified all of its elements (steps, input and output work products and roles). For example, the *Domain Knowledge* task comprises two steps that are performed by the *Domain Specialist*. This task generates a *list of sources of knowledge* (e.g. languages and examples of diagrams from the application domain) as output that will be used in the next task.

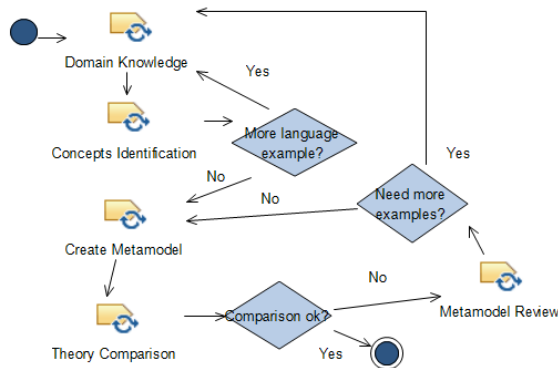


Figure 2. Conceptual Modeling workflow

This guide may be used in several domains. As our laboratory works with Model Driven Development [2] and model transformations, we used the design of a metamodel for the transformation domain as an example to guide explanation. In section 3(A) we briefly introduce the transformation domain and then in the following subsections we detail each one of the guide phases and tasks.

A. Designing a Metamodel for Transformations Domain

Model Driven Development (MDD) is a software development approach that makes intensive use of models instead of code. In MDD models are developed at a high abstraction level and transformed through a transformation chain until code. At the core of MDD is the transformation chain which encapsulates the mapping strategies to transform input into output models. The transformation chain comprises a

set of transformations responsible for automating/semi automating the MDD software development process [2].

Transformations receive models as input and generate models or texts as output [8]. Input and output models should conform to metamodels. The design of a transformation requires the definition of the relationships among elements of the source metamodel to elements of the target metamodel. A transformation itself may be specified as a model, called a model transformation model [21], which also should conform to a metamodel. In this scenario metamodels are necessary to: model the input and output models; develop the transformation chain (the relationships between source and target metamodel elements); and to design the model transformation metamodel.

In this paper we show the design of the Metamodel for Model Transformation (MMT) to illustrate our guide tasks. MMT is defined to support the development of model transformations at a high abstraction level. It comprises the necessary concepts for transformation specification and design independent of platform through a MDD approach to develop model transformations. So transformations code can be generated from the specification of transformation models.

B. Conceptual Modeling Phase

The main goal of the first phase of the guide, *Conceptual Modeling*, is to identify the relevant concepts of the domain. The result of this phase is the preliminary release of the metamodel. It consists of five tasks: *Domain Knowledge*; *Concepts Identification*; *Create Metamodel*; *Theory Comparison*; and *Metamodel Review*.

The first task (*Domain Knowledge*) consists of learning about the domain. Similar to the strategy used in [3] to identify metamodeling candidates for patterns, the most popular languages or some examples of applications designed in the domain should be selected.

Considering our example, the design of the transformation metamodel MMT, the languages initially selected were QVT (query / view / transformation) [10], because this is the OMG standard to design model transformations and ATL (Atlas Transformation Language) [11] due to its wide use in MDD projects to develop transformations.

In the second task, *Concepts Identification*, we should analyze the selected languages / application examples to identify the commonalities and specificities of the domain. The common concepts are then selected to be used in the construction of the metamodel. In the design of the MMT metamodel we had analyzed the constructors of the ATL and QVT (Relation) languages to find their commonalities and specificities. For example, in ATL a transformation is a *Module* comprised of *Rules*. There is one kind of rule, named *Matched Rule*, which is automatically executed when a source element matches a target element. Similarly, in QVT a *Transformation* comprises *Rules* that are specialized in *Relational Rules* for declarative definitions. The *Relational Rule* can be defined as a *Top Relation* to indicate that it must hold in order to be executed. Comparing the concept of transformation in these two languages, in MMT we defined both the *Transformation* and the *Relation* concepts and for the *Relation* we added an

attribute (*isRequired*) that indicates when the *Relation* must hold in a transformation execution.

In the third task, *Create Metamodel*, the previously selected concepts were organized as classes and their associations, generating the initial release of the metamodel. Attributes are also identified for the concepts.

The following task, *Theory Comparison*, consists of analyzing transformation theoretical concepts and comparing them to the concepts used in the initial release of the metamodel. Different theoretical approaches can be used in comparison, such as taxonomies and ontologies.

In the design of MMT we used the taxonomy presented in [9] as a reference to perform the comparison. This taxonomy classifies the concepts of transformation domain and its purpose is to address the essential characteristics of model transformations and existing languages and tools. Table 1 illustrates part of the comparison done.

TABLE I. PART OF THE TAXONOMY COMPARISON

Taxonomy [9]	Representation in MMT
Transformation type (Model transformation or Program transformation)	<i>Transformation</i> was specialized in: <i>M2M Transformation</i> for the model transformation type; <i>M2T Transformation</i> for the program transformation type
Endogenous x Exogenous transformation	Endogenous transformation: use the same metamodel on <i>SourceModel</i> and <i>TargetModel</i> associations; Exogenous transformation: <i>SourceModel</i> and <i>TargetModel</i> are different metamodels
Reuse (generic reuse, HOT, grouping, composition, decomposition)	Transformations are composed of other transformations (auto association on <i>M2M transformation</i>). It is possible to reuse existing transformation (combining them) to build new ones. It also allows the use of high order transformation (HOT)

The first column lists the taxonomy concepts and the second lists how MMT interprets these concepts. Cells in gray emphasize the concepts of MMT that were modified in order to suit the taxonomy. For example, in MMT the *Transformation* concept was specialized as *M2M Transformation* and *M2T Transformation* to support the two kinds of transformation modeling present in the taxonomy.

After the comparison, the *Metamodel Review* task was performed and some modifications were done in the metamodel. In our example, MMT, an association was added to the *Transformation* concept allowing transformation composition and reuse and the *Transformation* concept was specialized in *M2M Transformation* and *M2T Transformation* as partially shown in Fig. 3.

C. Metamodel Design Phase

The main goal of this phase is to organize the concepts defined for the initial structure of the metamodel. This phase comprises four tasks: *Structural Design*; *Constraint Definition*; *Functional Test*; and *metamodel Review*. A detailed release of the metamodel should be generated at the end of this phase.

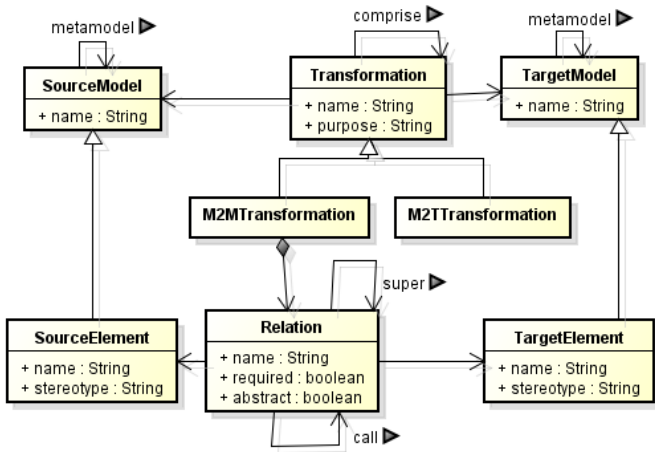


Figure 3. Part of MMT after conceptual modeling phase

The first task consists of the structural organization of the metamodel. Many kinds of strategies can be applied in order to structure the metamodel concepts, such as the use of packages to aggregate reusable concepts, the definition of general concepts to represent common attributes, etc. We recommend the adoption of the strategies proposed by [4]. These strategies guide metamodel developers in terms of: definition of packages to enable reuse of concepts; specification of association, e.g. how to define association member end features; specification of common attributes; how to create generalizations; definition of default values; when to use enumeration; and so on.

For the MMT metamodel many strategies from [4] were used. For example, we first used the strategy *Adding Abstraction Package* to group the constructors into two packages separating them in abstraction levels *MMTSpec* and *MMTDesign*. Then we used the strategies *Abstracting Common Attributes*, *Abstracting Common Associations* and *Generalizing Common Attributes* to identify constructors with the same attributes and/or associations and create a generalization for these common definitions (e.g. the *Model* concept was created to generalize *sourceModel* and *targetModel*), we defined the association end names and defined enumerations (Fig.4).

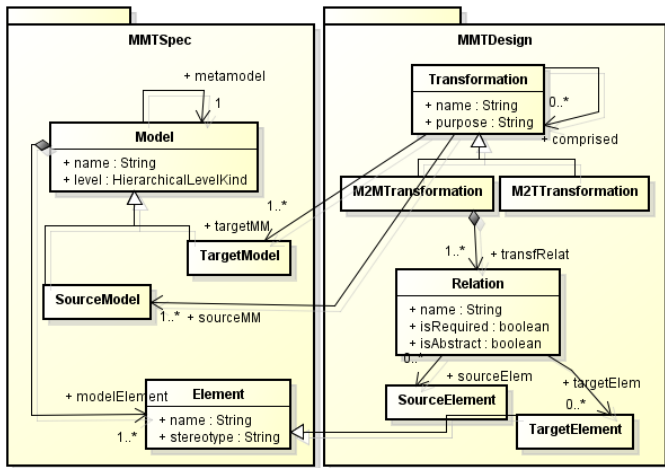


Figure 4. Part of MMT metamodel after Metamodel Design Phase

With the metamodel concepts defined and well-structured the next task, *Constraints Definition*, consists of the specification of the metamodel constraints using OCL (Object Constraint Language). For the MMT we defined constraints to specify model / metamodel conformance.

Although we had defined the concepts based on the available theory and have organized these concepts applying structural techniques, we had never used MMT to model a transformation yet. So, the last task of this phase, named *Functional Test*, consist in specify an instance of the metamodel and evaluate the effective use of the defined concepts and relationships in the instanced metamodel.

For the MMT *Functional Test* we instantiated the OO2RDBMS transformation which receives a class model as input and generates a logical data base model as output. Besides this we use UML diagrams stereotyped by MMT to visually model the transformation. The adoption of UML was a decision based on some premises: UML diagrams are well known by system developers; UML has a large number of tools to support the development tasks. The complete specification can be seen in [7].

After the functional test we observed that MMT concepts were almost sufficient to model the transformation. However, we observed a deficiency in the low level design of this transformation because we specified which elements of the source metamodel were transformed into elements of the target metamodel but we did not specify how this transformation should be done. As a result the metamodel was modified again to introduce the necessary elements for the lower level specification (*Metamodel Review* task).

D. Metamodel Validation Phase

The main goal of this phase is to evaluate the metamodel expressiveness in terms of coverage of the defined concepts. It comprises four tasks: *Validation Definition*; *Validation Execution*; *Validation Analysis* and *Metamodel Review*. The first task of this phase, *Validation Definition*, starts with the goal and the definition of the research questions. Any guideline related to software engineering experimentation can be used in this task, such as the guidelines presented in [23]. In the second task, the validation is performed (*Validation Execution* task) and according to the results (*Validation Analysis*) the metamodel can be modified (*Metamodel Review* task).

Regarding the design of the MMT metamodel we used a GQM template [22] (Fig. 5) to summarize goal definition and defined the following questions: (Q1) Do the MMT constructors sufficiently specify transformations written in ATL/QVT? (Q2) Is it necessary to add new constructors in MMT to enable the transformation specification written in ATL/QVT?

Analyze the MMT constructors
For the purpose of evaluating the metamodel expressiveness
With respect to coverage level
From the perspective of transformation developers
In the context of existing transformation developed in ATL/QVT languages

Figure 5. Experimental goal according to GQM template

The validation of MMT was executed over five months. During this period seven transformations that had already been developed in ATL / QVT language were specified using MMT. The transformations were selected from web repositories such as [24]. We performed the experiment in two stages (an initial test and the main experiment) where some dependent variables were measured, such as *specification completeness* and the *amount of used / new constructors*. After the validation we were able to conclude that MMT concepts covered most of the transformation specification, although some points for improvement were identified. For example, we observed that MMT could implement the concept of *transformation design pattern* proposed by [13] in order to simplify the specification.

Considering the results obtained, we performed the third task of the *Validation* phase, *Metamodel Review* and modified the metamodel (e.g. we added the *Pattern* concept in the MMTLowDesign).

During *Validation* phase, developers might observe the necessity of new attributes, associations or even new concepts that should be added to the metamodel. Therefore, the *Validation* phase can be done iteratively, instantiating the metamodel in different models, until developers observe that the amount of modifications decreases considerably. At this point the metamodel is considered stable enough for use.

IV. METAMODEL DESIGN GUIDE VALIDATION

Methods and processes for validation which involve humans are very challenging and they should be carried out in phases. Each phase should be an evolution from the previous one. So we first decided to evaluate the feasibility of the guide in driving developers in metamodels design. We followed the guidelines for software engineering experimentation presented in [23] and used GQM template [22] to define the goal of the experiment (Fig. 6).

Analyze feasibility of using the design metamodel guide
For the purpose of improving metamodel development
With respect to metamodel coverage and completeness
From the perspective of metamodel developers
In the context of model driven development

Figure 6. Goal definition according to GQM template

To reach our goal the following questions were defined:
 Q1: Does the guide help developers in metamodel definition?
 Q2: Does the guide improve the quality of the produced metamodel? In relation to the quality, we checked the coverage level of the defined concepts and the metamodel completeness. Accordingly, the following *null/alternative* hypotheses were formulated.

H1₀/H1: The use of the guide [*does not impact*]₀/[**decreases**] the participants difficulty in producing metamodels.

H2₀/H2: The use of the guide [*does not impact*]₀/[**improves**] the coverage of the concepts of the metamodels created by developers.

H3₀/H3: The use of the guide [*does not impact*]₀/[**improves**] the completeness of the class diagram created to represent the metamodel.

To evaluate the proposed guide we performed a controlled experiment to verify whether the guide improves metamodel development in building metamodels. Therefore we defined one independent variable, the *modeling method*, used to create the metamodel which varies across two groups: the *control group*, which developed metamodel in ad hoc way; and the *guide group*, which developed metamodel using our guide. The dependent variables are the *metamodel coverage*, the *metamodel completeness* and the *perceived difficulty* in metamodel development.

The experiment was performed over the period of 4 months divided in two phases: Initially a pilot study was performed and then the main experiment. The participants were undergraduate students (four students in the pilot study) and master degree students (twelve students in the main experiment). All of them had knowledge in MDD and process specification using languages such as SPEM. None of them had knowledge in metamodel design. Students were arranged in two groups according to the design method: the *control group* and the *guide group*. The data presented in the rest of this section relates to the main experiment.

As we had already developed a metamodel for the definition of MDD processes in our laboratory [12] we used this work as the problem domain in this experiment. To perform the experiment students received some examples of MDD processes and the SPEM specification. For the guide group we also gave the proposed guide and asked them to follow it. The experiment comprises three activities (related to our guide phases): *define metamodel concepts*, *specify metamodel structure*, *validate metamodel*. Students spent three days (one day for each activity) to design the metamodel, and each activity started for all students at the same time. We gave a maximum of two hours a day to perform each activity and measure how much time each student spent completing each task. At the end of the third day the students were supposed to have a metamodel representing MDD Process domain and an instantiated model conformed to this metamodel. Finally, participants answered a questionnaire that collected their perceptions about the difficulty of performing the tasks.

The produced metamodels were analyzed by our research group and compared to the reference metamodel, the metamodel that we had already developed for this domain. This comparison was based on a previously defined template and aims to evaluate metamodel coverage and completeness. The metamodel coverage was defined considering the amount of concepts identified by developers for the domain and attributes of each concept. The metamodel completeness was defined considering the amount of structural aspects specified in metamodels. We analyze the use of specializations, associations between concepts, the use of patterns to name variables, the use of enumerations and so on.

For the first activity, *define metamodel concepts*, we observed that the *control group* (who designed the metamodel in an ad hoc way) identified 66% of the concepts and the *guide group* (who used our guide) identified 87%. Nevertheless, the second group did it in greater detail (e.g. they also defined the attributes of each concept) so that the coverage rate was higher for the group that used our guide. For the second activity,

specify metamodel structure, we observed a big difference between the metamodels produced by the two groups. In the first group we noticed the absence of *specializations*, *enumerations* and *associations end role* definitions which make these metamodels more verbose and difficult to understand. As a result the completeness rate for the *control group* was 45% while for the *guide group* it was 81%. The third activity, *validate metamodel*, was not so helpful for our evaluation because all the participants could instantiate a model conforms to their metamodel (it differs on the level of detail of each instance according to the coverage/completeness of the metamodel). Analyzing the perceiving difficulty reported by the participants we noticed that metamodel design is still a challenge due to the high level of abstraction needed in definitions. We did not observed any big difference between the two groups related to the time spent by each participant on performing the three tasks in the experiment. However we observed that the metamodel guide led to metamodels with a high level of coverage and completeness.

To decrease any threat to validity some strategies were adopted. With regard to participant knowledge we checked that they were familiar with the MDD approach (and metamodeling) as they were students doing an MDD course or they had already participated in MDD projects. We used randomization to assign participants to each group and prevented communication between them. As far as the tasks were concerned we gave all the participants the same amount of time to perform them. Besides this the domain might be another threat so we tried to choose a domain familiar to software engineers.

Empirical assessment usually takes into account the amount of data collected from the subjects. However, in the case of a guide validation it is difficult to involve a great number of people in case studies. A case study rather than a rigorous experiment was the most suitable choice. We know that the study results are limited and do not provide statistical evidence to support general conclusions. However, we believe that it can be considered an initial step in future case studies to be performed in order to observe other aspects.

V. CONCLUSION

This paper reported our experience in metamodel design through the definition of the MMT metamodel. From this experience and some expertise in the domain of MDD and software engineering we defined a guide for metamodel developers that we believe can be used as a base for a metamodel development methodology.

Two main difficulties were encountered in this work: the metamodel conceptual modeling and the validation. To address the first one we used a taxonomy and specific languages (e.g. ATL) to identify the relevant concepts of the transformation domain. Validating a metamodel is quite different from validating a piece of software. Metamodel validation requires instantiation examples. As a result alternatives were used in validation (e.g. reverse engineering techniques) until we considered the metamodel was stable enough.

We believe that the guide to develop metamodels generated by our experience can be adapted and evolved to be used in the

design of other kinds of metamodels other than transformation domains. We are now working on this generalization and on case study scenarios to achieve this goal.

REFERENCES

- [1] S. Mellor, A. Clark, T. Futagami "Model Driven Development" IEEE Software, 2003
- [2] T. Stahl, M. Volter, J. Bettin, A. Haase, S. Helsen foreword by K. Czarnecki "Model-Driven Software Development" Wiley, 2006.
- [3] H. Cho, J. Gray "Design Patterns for Metamodels" SPLASH'11 workshops, Portland, Oregon, USA, October, 2011.
- [4] A. Vieira, F. Ramalho "Identifying Guidelines for Constructing Metamodels" In: III Brazilian Workshop on Model-Driven Software Development, Natal, 2012.
- [5] R.S.P. Maciel, R.A. Gomes, A.P. Magalhaes, B. Silva "Supporting model-driven development using a process-centered software engineering environment." ASE, v1, p1, 2013.
- [6] A.P. Magalhaes, J. David, R.S.P. Maciel "Modden: an integrated approach for Model Driven Development and Software Product Line Processes" in 5th SBCAR, São Paulo, 2011.
- [7] A.P. Magalhaes, A. Andrade, R.S.P. Maciel "MTP : Model Transformation Profile" In : 7th SBCARS, Brasilia, 2013.
- [8] M. Brambilla, J. Cabot, M. Wimmer "Model-Driven Software Engineering in Practice" Morgan & Claypool Publishers, 2012.
- [9] T. Mens, K. Czarnecki, P. Van Gorp "A Taxonomy of Model Transformation" Dagstuhl Seminar Proceedings 04101, 2005.
- [10] QVT specification - <http://www.omg.org/spec/QVT/1.0/PDF/>
- [11] ATL Project - <http://www.eclipse.org/m2m/atl/>
- [12] Maciel, R. S. P. ; Magalhães, A. P. ; "An Integrated Approach for Model Driven Process Modeling and Enactment." In: SBES , Fortaleza, Brazil 2009
- [13] Iacob, M.; Steen, M.; Heerink, L. "Reusable Model Transformation Pattern." In 3M4EC'08, pages 1-10, 2008.
- [14] Fowler, M. "Domain Specific Languages." Addison Wesley, 2011.
- [15] Kleijnen, J.; Sargent, R. "A methodology for fitting and validating metamodels in simulation." European Journal of Operational Research, pp. 14-29, 2000.
- [16] Cuadrado, J.; Molina, J. "Building Domain-Specific Languages for Model-Driven Development", Software IEEE 24.5, pp.48-55, 2007.
- [17] Luoma, J.; Kelly, S.; Tolvanen, J. "Defining Domain-specific modeling languages: collect experiences." 4th Workshop on DSM, 2004.
- [18] Karsai, G.; Krahn, H.; Pinkernell, C.; Rumpe, B.; Shindler, M.; Volkel, S. "Design Guidelines for Domain Specific Languages." In proceedings of Domain-Specific Modeling, 2009
- [19] Bak, K.; Zayan, D.; Czarnecki, K.; Wasowski, A.; Rayside, D. "Example-Driven Modeling. Model=Abstraction+Example." ICSE, 2013, San Francisco, USA.
- [20] Mernik, M.; Sloane, A. "When and how to develop Domain Specific Languages." ACM Computing Surveys, Vol 37, Nr.4, pp.316-344, 2005.
- [21] Bézivin, Jean et al. "Model Transformations? Transformation Models?" Springer-Verlag Berlin Heidelberg, 2006
- [22] Solingen, R. Basili, V.; Caldiera, G.; Rombach, H.D. Goal Question Metric (GQM) Approach. John Wiley & Sons. Inc., 2002.
- [23] Wohlin, C. Aurum, A. Towards a decision-making structure for selecting a research design in empirical software Engineering. Empir Software Eng DOI 10.1007/s 10664-014-9319-7. Springer, 2014
- [24] SimpleGT, available in "<http://soft.vub.ac.be/viewvc/SimpleGT/>
- [25] Mellor, S.; Clark, A.; Futagami, T. "Model Driven Development" IEEE Software, 2003
- [26] Cicchetti A.; Ruscio, D.; Kolovos, D.S.; Pierantonio, A. A Test-driven approach for metamodel development. Chapter of the book Emerging Technologies for Evolution and Maintenance of Software Models, p. 319-342, IGI Global, 2012.
- [27] Lara, J.; Guerra, E. Generic meta-modelling concepts, templates and mixin layers. Models, 2010.

Finding and Emulating Keyboard, Mouse, and Touch Interactions and Gestures while Crawling RIA's

Frederik H. Nakstad, Hironori Washizaki, Yoshiaki Fukazawa
Department of Fundamental Science and Engineering
Waseda University
Tokyo, Japan
frederik@fuji.waseda.jp

Abstract—Crawling JavaScript heavy Rich Internet Applications has been a hot topic in recent years, giving us automated tools for indexing content, test generation, and security- and accessibility evaluation to mention a few examples. However, existing crawling techniques tend to ignore user interactions beyond mouse clicking, and therefore often fail to consider potential mouse, keyboard and touch interactions. We propose a new technique for finding and exercising mouse, keyboard, and touch interactions when crawling highly interactive JavaScript-based websites by analyzing and exercising event handlers registered in the DOM. A basic form of gesture emulation is employed to find states accessible via swiping and tapping. Testing the tool against 6 well-known gesture libraries and 5 actual RIA's, we find that the technique discovers many states and transitions resulting from such interactions. Our findings indicate the technique could be useful for automatic test generation, error discovery, and accessibility evaluation, especially for mobile web applications with advanced interaction options.

Keywords—crawling; gesture emulation; event handler analysis; RIA

I. INTRODUCTION

Web applications have become more and more advanced over the past few years with the maturation and increased adoption of HTML5 and its related API's. As a result of this, the amount and complexity of JavaScript code on the client-side has grown, giving us a different breed of web application capable of much more advanced functionality and richer user interaction models than before. Compounding this is the advent of the smartphone, which has popularized touch screens and made various gesture interactions part of most people's technical vocabulary. A recent report finds that users in the US now spend more time consuming media using their mobile and tablet devices than desktop computers [8].

This rise in client-side complexity coupled with the dynamic nature of JavaScript has introduced many challenges in how to reliably crawl such web applications. While traditional websites rely on anchor tags and buttons for navigation and are static in terms of content on the client-side, new JavaScript-reliant RIA's can change dynamically and drastically as the result of JavaScript manipulating the DOM of the page without the need of a round-trip to the server for new HTML. In papers such as [1, 7] methods for automatic crawling of JavaScript reliant RIA's have been introduced.

These methods have spawned a variety of applications to automate beneficial tasks such as indexing for search engines [1], accessibility and usability evaluation [6, 16], automatic test generation [3, 11, 12, 13], regression testing [4], and security testing [5] to mention some.

One aspect oft neglected is the fact that this new breed of web applications are capable of very advanced interactions. Existing research focuses heavily on simple interactions initiated by mouse clicks, usually ignoring other keyboard, mouse, and touch events, not to mention gestures. Another common assumption is that most state transitions can be reached by interactions with `<a>`, `<button>`, and `<input>` elements when choosing candidate interactions as often done by empirical studies such as [10]. This might be okay for web sites containing simple interaction models on this small subset of elements, but we believe many states and transitions could be missed for RIA's with more advanced UI and mobile web applications.

In this paper we propose a set of extensions to Crawljax with functionality enabling it to detect the event handlers available in each state. These detected event handlers are then used as a basis for identifying new candidate interactions. The candidate interactions are exercised using programmatic event construction and gesture emulation, leading to an increase in discovered states and transitions.

The following research questions are addressed:

- **RQ1.** How comprehensively and efficiently can our technique capture and perform gesture interactions?
- **RQ2.** How often do various keyboard, mouse, and touch events lead to new states in modern RIA's, and which event types are more likely to induce new states?
- **RQ3.** What DOM elements are more likely to be targets for interactions leading to new state transitions?

The following contributions are offered:

- A technique for discovering all event handlers of a state when crawling websites.
- A set of extensions to Crawljax enabling it to identify candidate transitions based on event handler analysis and emulate simple event dispatches as well as tap and

swipe gestures. Henceforth this modified version of Crawljax is referred to as mobCrawler¹.

- Evaluation of mobCrawler's ability in emulating common gestures and interactions by testing against 6 of the most popular gesture libraries for JavaScript available.
- A case study against 5 modern RIA's with advanced interaction models evaluating the efficiency and usefulness of event handler analysis when crawling advanced RIA's.

II. BACKGROUND

A. Crawling RIA's and Crawljax

Crawling websites used to be relatively less complex than it is today. For traditional websites the content of each page is static after it has been loaded in the browser, and other static pages were loaded by following anchor or button elements. In modern rich Internet applications using JavaScript things are more complex. Changes to the DOM can come as the result of asynchronous HTTP requests loading new content, or event handlers and timeout events firing custom code. Additionally, the dynamic nature of the JavaScript language itself can provide flexibility as well as unintended states and side effects [2].

Crawling such applications is usually done by loading a starting page, its DOM recorded as the initial state, and then automatically exploring the various interactions possible on the page to elicit changes in the DOM. Each interaction causing a change is recorded as a transition, and each modified DOM is recorded as a new state.

One of the most prominent tools fulfilling this purpose is Crawljax. Crawljax is a highly customizable crawler aimed at JavaScript-heavy web sites. It performs a depth-first search of the target web site using the Selenium Web Driver to control the browser, and has many customization options for things such as setting crawl depth, state abstraction comparators, and crawl rules for ignoring certain links. It performs a depth-first search trying to detect as many states and transitions as possible within the constraints specified by the user. Due to this open and customizable nature we implement our technique as a set of modifications and extensions to Crawljax.

B. Event Handler Registration

When adding event handlers to a web page, there are three options available through the browser API:

1. Programmatically use a target elements `addEventListener()` function
2. Programmatically use a target elements `on[eventType]` attribute, `[eventType]` indicating the type of event handler

3. Declare a `on[eventType]` attribute on the target elements HTML tag, `[eventType]` indicating type of event handler

There are countless libraries offering other API's to attach event handlers, but they all eventually resolve to one of these standardized, "native" API calls offered by the browser. Using any of these options, developers are able to attach custom code to be fired when one of a multitude of event types is performed on the given element. The code in these event handlers may manipulate the DOM, potentially eliciting a new state. Gestures are usually developed as a sequence of related event handlers on event types such as `touchstart`, `touchmove`, and `touchend`, analyzing each event sequence's properties to determine what gesture was performed.

Though a common pattern is to attach event handlers once the `DOMContentLoaded` event of the browser has fired as part of the page's lifecycle, developers can technically add events as soon as JavaScript is being evaluated by the browser. New event handlers can also be attached at any point after page load, as part of other event handlers being exercised.

C. Mobile Devices

According to [9] web browsing on mobile and tablet devices has increased rapidly the last few years, and together constitutes 37.48% of all usage. As mobile navigation of the web increases, developers may also want to adapt their website to take advantage of touch screen input and gestures. This could mean that many states and transitions not reachable by simple mouse clicks anymore. Previous techniques may miss important states relying on touch, mouse, and keyboard interactions, especially for mobile web applications relying on gestures. We believe actually exercising such interactions will help in finding new states and transitions whether it is for indexing content, automatically evaluating accessibility, or employing automated testing approaches for the target web application.

There has been a huge increase in use of the web from smartphones [9], and as a result of this many web applications also offer an especially tailored mobile version. These mobile versions often take advantage of the devices touch screen, and may implement certain navigational interactions using touch and gestures. It therefore seems likely to us that such web applications may contain states and transitions only reachable through touch and gesture interaction.

```
<div id="gallery">
  
  <label class="caption">Mt. Fuji</label>
</div>

$('#gallery').swipeLeft(function() {
  // Load new image and caption via AJAX
  ...
});
```

Figure 1. Gesture Handler Registration

¹ <https://github.com/fnakstad/mobcrawler>

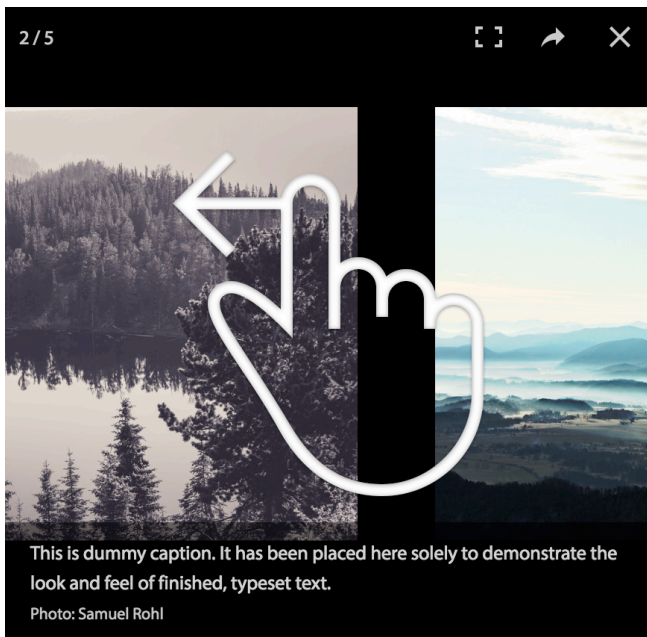


Figure 2. Gallery with swipe interactions

D. Motivating Example

For our motivating example we consider a photo gallery component embedded on a web site containing an image and some descriptive text as pictured in Figure 1 and 2. In order to load the next picture and text caption in the gallery you have to swipe left or right. These swipe interactions would likely be implemented by attaching the desired gesture type to a target DOM element as seen in the code sample. Since existing crawl techniques don't attempt to execute such advanced interaction events, the states loaded by performing these swipe gestures would not be found, leaving a good chunk of application functionality and content unexplored. This kind of interaction pattern has become quite common and can often be seen in photo galleries, carousels for news stories, especially featured sales products, and so on.

III. EVENT HANDLER ANALYSIS AND GESTURE EMULATION

A. Detecting Registered Event Handlers for a State

In order to find out what event handlers have been registered in any given state it is necessary to instrument the native `addEventListener()` function implementation to keep track of all event registrations performed by the target web site's JavaScript code. It is important that our instrumentation code is performed before any other client-side code in order to catch all events programmatically added to the page. This means we need to inject this instrumentation module at the very top of the documents `<head>` tag before the page is loaded in the browser. This approach ensures that any event handlers attached during page load as well as dynamically when crawling to another state from initial page load will be detected by our tool.

```
var original =
  EventTarget.prototype.addEventListener;
EventTarget.prototype.addEventListener =
function() {
  var type = arguments[0];
  addEventHandler({
    type: type,
    xpath: getElementXPath(this),
  });
  original.apply(this, arguments);
};

window.addEventListener('load', function(){
  walkSubtree(document);
}, false);
```

Figure 3. Event Handler Registration Detection

There may also be event handlers attached directly via `on[eventType]` attributes on the DOM elements. These event registrations are not processed through `addEventListener()`, so we need to handle them separately. After the page is loaded we walk the entire DOM tree, polling each element for any such event handler registrations.

As seen in the code snippet in Figure 3, The `addEventHandler()` function will check whether the event type is of a type we are interested in, which can be configured in the script, and if so add it to a list. Once the DOM is loaded, the script will call the `walkSubtree()` function on the document object, which will recursively traverse the entire DOM tree looking for event handlers attached via the `on[eventType]` attribute, and add them via `addEventHandler()`. This means we can effectively monitor any and all event handler registrations performed on the page.

B. Injecting Script via a MITM Proxy

Figure 4 shows the structure of `mobCrawler`. Neither the browser itself nor the Selenium WebDriver used by `mobCrawler` to control the browser offers a way to manipulate the HTML before it is evaluated. Therefore we inject the instrumentation script using a faux man-in-the-middle attack via a proxy server.

This proxy server is implemented using the well-known `mitmproxy` [14] program, and run with a custom script. The proxy server sniffs for any relayed HTML content, intercepts it, and modifies the `<head>` tag to inject our JavaScript module. The injected JavaScript module then attaches one non-intrusive JavaScript object on the window object, which can be configured to listen for event handler registrations of any specified event types. `mobCrawler` can communicate with this module via JavaScript operations executed in the browser to fetch the current event handlers for a state as well as exercise various interactions on the web page. Using a man-in-the-middle proxy server poses some problems with regards to the browser denying 3rd party SSL certificates for certain sites or forbidding functionality which might be restricted due to CORS security policies. A preferable option would be to use the WebDriver or web browser itself to inject this script into the document before evaluating it, however this is not supported as of today. We configure `mobCrawler` to use this proxy server when crawling.

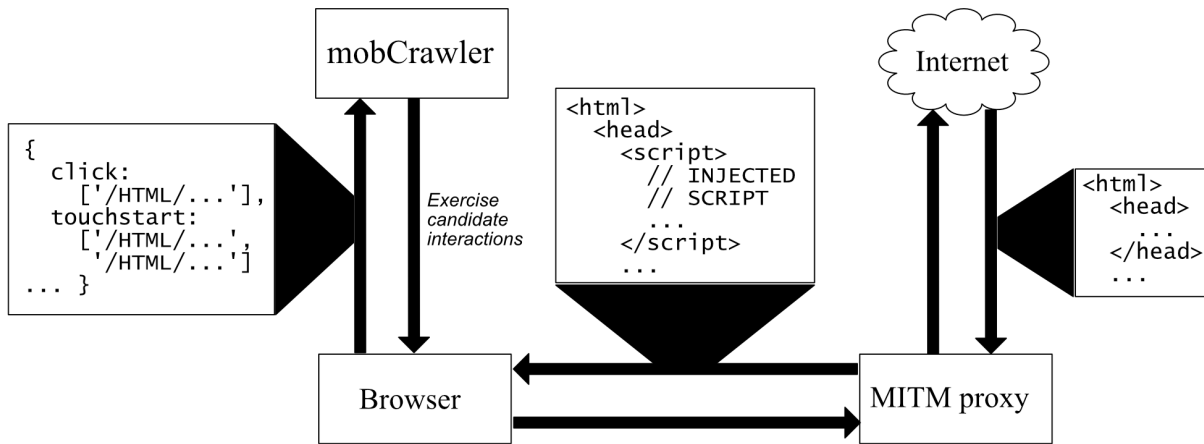


Figure 4. Structure of mobCrawler

TABLE I. GESTURE EMULATION

Interaction	Description	Fire On
Event dispatch	Create and dispatch an event to the target element	click, dblclick, mouseover, mouseout, keydown, keyup, keypress
Mouse swipe	Emulates a mouse swipe in a given direction (left, right, up or down), by creating and dispatching a series of mousedown, mousemove, and mouseup events.	mousedown
Touch swipe	Emulates a touch swipe in a given direction (left, right, up or down), by creating and dispatching a series of touchstart, touchmove, and touchend events.	touchstart
Tap	Emulates a tap by dispatching touchstart and touchend events with a small timeout.	touchstart
Double tap	Two tap events in quick succession.	touchstart
Tap hold	Same as a tap but with a longer timeout between the touchstart and touchend events.	touchstart

C. Emulating Browser Interactions

mobCrawler can now fetch all registered event handlers registered in a state by calling our injected JavaScript module. This list of event handler registrations will in turn form the basis for what candidate elements and interactions we choose to perform. Though we can possibly detect and emulate all possible browser event types, we choose to focus on mouse, keyboard, and touch events. For some of these events, such as mouseover, mouseout or keydown it is sufficient to create and dispatch a corresponding event object to the target element. These events are created via the JavaScript module injected into the page, and will programmatically construct events of the desired type mirroring the W3 specifications in [17]. The parameters set will be based on the target element in question and what the desired interaction specified is, then finally dispatched to the target element in question.

For keyboard events there is a big span of keys that can be set to be activated. In our simulations we use the somewhat naïve approach of always setting the key being pressed to the

character ‘e’. It seems like many more states can be elicited with a more advanced strategy of when to use what characters. Many websites include various keyboard shortcuts which could increase the number of found transitions and states if exercised. However we do not consider that beyond our rather simple approach in this paper.

Additionally, we emulate certain common gestures using mouse and touch events as specified in Table I. Variations of tap gestures are only considered for touch, while swipe gestures are also implemented for mouse. In this paper we only consider single-touch gestures, ignoring more advanced gestures involving more fingers. We do this since we find it less likely they will induce meaningful states and to reduce the number of candidate interactions performed.

Swipe gestures can be emulated in 4 different directions, and are calculated by starting at the center of the target element, then rapidly being moved 400 pixels in the given direction. Candidate interactions are created based on the event handler registrations detected in the current state. This means that if a touchstart event handler is found on an element, a total of 7 candidate interactions will be attempted: swipe up, swipe down, swipe left, swipe right, tap, double tap and tap hold. This can lead to a rapid increase in candidate interactions, but can be customized to add only a subset of these candidate interactions. For example, we could instruct mobCrawler to only try swipe right and single tap interactions for elements containing touchstart event handler registrations if that is desired.

Using this approach we can see how the problem posed in the motivating example can be solved. The JavaScript module injected into the browser via the MITM proxy will detect a touchstart event registered on the gallery element, and record it in a list of event handler registrations. Once the page has loaded mobCrawler will fetch this list of event handler registrations, and process them in turn. When it reaches the touchstart event registered on the gallery element, it will attempt various swipe and tap gestures as described in Table I to see if the action changes the state of the DOM. Once it tries swiping left and swiping right the content inside the gallery should change, and mobCrawler will register these new transitions and states in the state graph.

TABLE II. GESTURE SUPPORT

Library	Tap	Double tap	Tap hold	Swipe			
				Left	Right	Up	Down
Hammer.js	✓	✓	✓	✓	✓	✓	✓
Quo.js	✓	✗	✗	✗	✗	✗	✗
dojox.gesture	✓	✓	✓	✓	✓	✓	✓
touchSwipe	✓	✓	✓	✓	✓	✓	✓
Touchy	✗	✗	✗	✗	✗	✗	✗
jGestures	✓	N/A	N/A	✓	✓	✓	✓

D. Mobile Websites

It has become increasingly common for websites to offer a separate mobile version, which can be very different from the plain “desktop” version. In many cases the mobile version is a completely separate implementation from the desktop version to create a user experience entirely tailor made for mobile and tablet devices. The most common technique used by websites to differentiate between mobile and desktop users is user agent sniffing, which will parse the user-agent string in the initial HTTP request, and then serve back either the mobile or desktop version of the website depending on the device reported in the user agent string.

We add functionality to mobCrawler which enables the user to imitate a mobile device by overriding the user-agent string sent in HTTP requests to the webserver of the site we are crawling. If the user wants to imitate a mobile device, this string can be set to the user agent string reported by a browser built for the iOS or Android platforms. Some gesture libraries and websites will also offer different functionality based on whether the browser reports the user’s device as being touch enabled or not. For such cases, we inject a polyfill spoofing that we support touch even if we are crawling using a non-touch device.

IV. EVALUATION

A. Gesture Detection and Emulation (for RQ1)

To evaluate the effectiveness of emulating gestures, we set up an experiment where we task mobCrawler with finding gestures as implemented by various popular JavaScript gesture libraries. Though it is possible to create gestures from scratch just by using the native touch and mouse events, it seems likely to assume that the majority of developers will rely on a gesture library to make their jobs easier. This also gives us the opportunity to investigate many different implementation approaches, and seeing if our tool can handle them.

We create a basic website for each library with a <div> element on which we register gesture interactions using the various libraries. Though many of the libraries offers a big set of gestures we focus only on simple single-point gestures as described in Table I.

As can be seen in Table II, the approach was able to handle the gesture interactions of most libraries flawlessly with two exceptions. Both Quo.js and Touchy create and register custom event types, which they use when registering their gestures. Since our code instrumenting `addEventListener()` registrations

was configured to only look for a predefined set of standard event types it didn’t pick up on these non-standard event type registrations. However, mobCrawler can be configured to also look for these custom events, and fire corresponding gesture interactions on them to expand library support.

The majority of gesture libraries seem to use the native touchstart and mousedown events, and in these cases our tool was able to exercise all the transitions and find all the given states. Note that jGestures did not have API’s for registering doubletap and tap hold gestures, and so those gestures are not considered. Hammer.js adds both mouse and touch handlers for gestures, while Quo.js and dojox.gesture actually checks mouse and touch capabilities of the device before registering the appropriate event handlers.

A drawback of the approach is the large number of candidate interactions which have to be exercised in order to find valid states. Since we can’t make any assumptions about what gesture interaction is required to elicit a new state, we resort to exercising all of them. For example, upon finding an element with a touchstart event handler attached, we create candidate elements for all 3 tap gestures as well as swipes in 4 different directions. This might lead to a high recall of new states, but precision is lacking and can lead to a lot of failed candidate interactions. Reducing the candidate gesture interactions to attempt could help reduce this number drastically, e.g. by just focusing on right swipes and single taps.

RQ1. How comprehensively and efficiently can our technique capture and perform gesture interactions?

To answer RQ1 we conclude that our approach can successfully handle the majority of ways to implement gestures in the browser, though it might be necessary with some extra configuration to support specific libraries using custom event types. Additionally, to increase precision of state and transition discovery developers may want to minimize what gesture interactions are attempted when crawling a site.

B. Case Study (for RQ2 and RQ3)

We proceeded to test out this tool on 6 actual websites to evaluate the remaining two research questions. The target websites were chosen due to having advanced GUI’s, being tailored for mobile, and being crawlable through the MITM proxy. C1 is an interactive application for manipulating animations, while C2 implements a MS Paint clone in the web browser. C3 is a web application for displaying presentations. C4 and C5 are informational sites customized for smartphone devices. We kept the level of state abstraction low, trying to consider all changes introduced to the DOM. However, in order to weed out false positives we do instruct mobCrawler to ignore ads and automatically created hash values on a site-by-site basis. Maximum crawl times are set to 2 hours, with a crawl depth of 2. We also instruct the browser to not accept any cookies.

TABLE III. CASE STUDY TARGET SITES

Case	URL	Mobile UA
C1	bomomo.com	No
C2	muro.deviantart.com	No
C3	slides.com/andreylisin/omaha-server#/6/1	Yes
C4	m.weather.com/weather/today/JAXX0085:1:JA	Yes
C5	touch.toyota.com/index.html	Yes

TABLE IV. CRAWL RESULTS

	C1	C2	C3	C4	C5
States, "click"	22	60	18	60	50
States, other	42	124	61	7	38
Transitions, "click"	22	60	23	61	100
Transitions, other	42	124	198	7	108
Transitions, <a> + <button>	4	16	0	29	49
Transitions, other elements	60	175	221	39	159

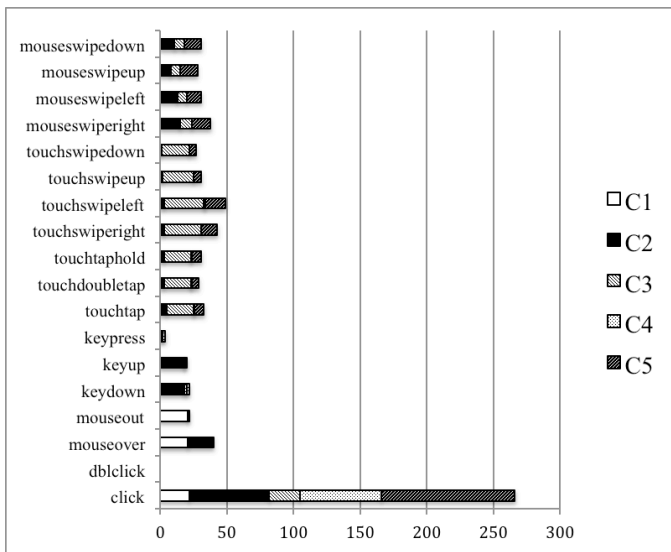


Figure 5. Event Type Distribution for State Transitions

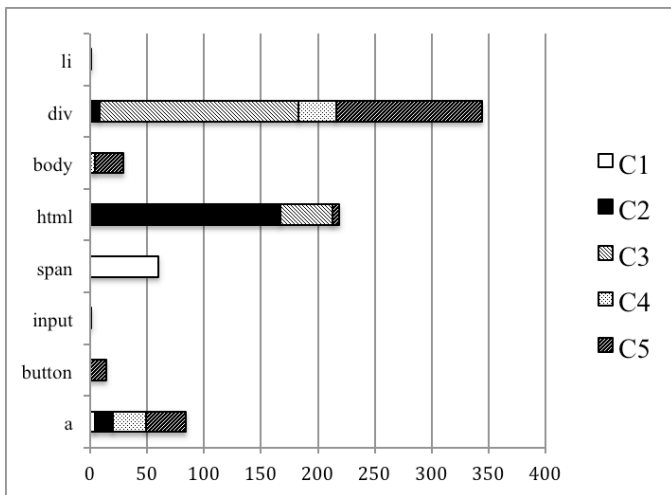


Figure 6. Element Type Distribution for State Transitions

As can be seen in Table IV, all the target sites have a large number of states and transitions being reached by exercising interactions other than “click”. In C1 a lot of the menu buttons’ CSS styling are changed as part of mouseover and mouseout events. C2 mirrors C1 in that most of the non-“click” induced states are often differentiated by changes in styling or visibility of elements. There are also a number of keyboard shortcuts to quickly select given menu options.

C3 depends on swipe gestures to switch between various slides. The high number of transitions can be explained by the fact that a lot of states can be reached by multiple interactions. For example, moving to the next slide in the deck can be accomplished by swiping right or using a button in the control panel. C4 and C5 are examples of more standard mobile web applications, and we can see that most states can be found using the “click” interaction. However, both sites contain image galleries or sections which can be swiped to change the contained content, similar to our motivating example. C5 has a huge number of transitions due to multiple interactions reaching the same state, as was also observed in C3.

Looking at Figure 5, we can see that “click” events still, unsurprisingly, still make up the major amount of transitions when crawling, though a considerable amount of transitions use other event types. In C3-C5 there are considerably many touch transitions, though many lead to the same state. Swiping horizontally seems to be more fruitful than vertically. Both the desktop sites, C1 and C2, have a fair amount of mouse and keyboard events. Though not inducing a state change we observed the swipes were also able to change what was drawn on the <canvas> element.

RQ2. How often do various keyboard, mouse, and touch events lead to new state transitions in modern RIA’s, and which event types are more likely to induce new states?

To answer RQ2, we conclude that swipe gesture transitions are very common for mobile web applications and can induce new and meaningful states. The huge number of transitions leading to the same state might be unwanted if crawling with the aim of discovering content, but may be desirable if trying to generate a comprehensive test suite or detecting errors in a web application. For desktop sites, both keyboard and mouse events are found to be common for advanced web applications with advanced GUI’s.

Considering the amount of non-“click” transitions in C2, C3, and C5, it seems they are too numerous to ignore. Though not all applications rely heavily on these kinds of transitions, such as C4, the number can be too high to simply ignore. As evidenced by the image galleries in C5 and slides in C3, content crawlers may be able to increase the amount of data found if they attempt to emulate swipe gestures on mobile websites. When it comes to automatically generating test suites for web applications, the developers may get a higher test coverage by instructing the crawler to attempt gesture types they know are present in the application.

Figure 6 depicts what elements were interacted with to find state transitions. As can be seen, this varies depending on the specific website, with C1 relying heavily on elements which it uses for its menu buttons. C2, C3 and C5 have large

numbers of event handlers attached to the `<html>` or `<body>` element. Many of these transitions are gestures which can be performed anywhere in the visible document. `<a>` and `<div>` elements are in consistent use in all the web sites. However `<a>` and `<button>` elements are not nearly as prominent as the `<div>` element in terms of inducing transitions.

RQ3. What DOM elements are more likely to be targets for interactions leading to new state transitions?

To answer RQ3 we can conclude that element types employed as interaction targets can differ widely depending on the website. `<a>` and `<div>` elements seem to be the most common interaction targets to use for transitioning between states. Web applications using gesture actions applied to the entire documents surface area will likely have interaction targets on the `<html>` or `<body>` element. The results indicate that event handler analysis is a more comprehensive and precise way of finding possible transitions in a state than exhaustively trying out a predetermined list of element types in each state.

The most obvious takeaway is that the types of elements used for inducing transitions are too varied from site to site, to rely on a small subset when targeting a large variety of web applications. This could be useful when performing empirical studies on a large host of web applications by way of crawling. By combining event handler analysis along with a somewhat selective set of gesture interactions to try out, the resulting state graphs could be richer and more representative of their respective web applications.

V. USAGE

Our findings indicate our technique can help augment existing tools that are automatically exploring web applications by increasing the number of found transitions. If our case study is a good indicator of a more general trend in web applications, there are many web sites containing advanced interactions which can be modeled better than with current crawling techniques. A fitting example would be content crawling mobile web applications relying on swiping to load new content. If a crawler can't emulate a user swiping on the given GUI element, the content loaded as a result may be left unexplored and unindexed.

Another area in which our technique seemingly can help is increasing code coverage in automatically created test suites for web applications relying on advanced interactions. This is well illustrated by C3 in our case study, which is a web application for browsing slide presentations in a mobile browser. When creating an automatic test suite for this application, it is reasonable to assume the developer also wants to generate test cases for the various gestures involved in manipulating the slides. By emulating these gestures we could trigger event handlers not triggered by traditional crawling techniques and thus increase the total code coverage of the web application. Possibly this could also lead to finding a number of unexpected errors and event sequences, and should generally increase the quality of the test suite.

For many of the same reasons stated in the previous two examples, we believe our approach would be beneficial for

various automatic evaluation techniques of rich web applications. Several approaches to accessibility and security evaluation rely on automatically exercising the GUI to find states and transitions as evidenced in [6] and [5] respectively. It seems reasonable to assume these approaches would benefit from our technique as they would be able to explore a larger number of transitions initiated by non-“click” interactions. Once again it seems like mobile web applications especially would benefit from this.

VI. RELATED WORK

Though Crawljax[1] can be configured to initiate simple click events on any kind of element, it relies on exhaustively exercising all such elements on a page rather than finding only the elements with “click” event handlers attached on them. The support for exercising different kinds of event types is minimal and focuses on simple mouse clicks. Our extensions allow it to find all desired events registered on a page via event handler analysis, and exercise simple even dispatches as well as single-point gestures.

There has been performed some previous research utilizing event handlers as evidenced by tools such as ARTEMIS from [3] and FEEDEX from [12]. These tools seem to analyze event handler registrations by using a modified version of the Rhino JavaScript interpreter. They use the event handlers as input in prioritizing what crawl action to take next. [18] introduces a symbolic execution framework for finding security vulnerabilities in web applications. It utilizes event handler analysis and can perform simple event dispatches, but does not attempt to emulate gestures. [15] introduces a static analysis approach to creating finite state machines from source code by analyzing event handlers. In contrast to our tool, none of these approaches try to find and emulate more complex gestures, and most of them, except [18], also seem to ignore simple event dispatches.

A static approach to analyzing interactions in JavaScript applications can be found in [23] which models web applications as finite state machines. However, it focuses on verifying interaction invariants by way of model checking as well as mutation analysis, while we are following a dynamic approach of modeling web applications as state diagrams for purposes such as test suite generation and content indexing. Related papers [22, 24] use the same approach, and target mutation analysis and testing AJAX and RIA's.

There has also been performed related research in the native mobile application space. [19] targets Android applications, and introduces a technique in which they first analyze event handlers, and then generate and exercise different sequences of events to reach hard-to-find states and transitions. [20] and [21] automatically analyzes event handlers in order to create and exercise events to find GUI bugs or create test suites for Android applications. While all these researches focus on apps on the Android platform, our research is focused on RIA's on the web.

VII. CONCLUSION

We have introduced a new technique for finding and exercising mouse, keyboard, and touch candidate interactions when crawling web applications, as well as a case study investigating its performance in modern RIA's. The technique we proposed seems like a good fit for automatic crawling of web applications with advanced interaction models, especially mobile. Automatic test generation and error detection, as well as automatic solutions for evaluating accessibility and usability seem like good use cases.

In the future we would like to replace the script injection mechanism with something less intrusive than a MITM proxy, as well as add a default configuration supporting the custom event types of all major gesture libraries. In addition, we believe it would be fruitful to do a bigger case study involving a number of the most popular mobile web applications to further investigate the applicability of the approach. This would give us a better idea of the robustness of the approach and whether the patterns observed in our limited case study holds on a larger scale.

REFERENCES

- [1] A. Mesbah, E. Bozdog, and A. van Deursen. "Crawling Ajax by inferring user interface state changes," Eighth International Conference on Web Engineering, 2008, pp. 122-134. IEEE.
- [2] G. Richards, S. Lebresne, B. Burg, and J. Vitek. "An analysis of the dynamic behavior of JavaScript programs," Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation, 2010, pp. 1-12. ACM.
- [3] S. Artzi, J. Dolby, S. H. Jensen, A. Moller, and F. Tip. "A framework for automated testing of javascript web applications," 33rd International Conference on Software Engineering, 2011, pp. 571-580. IEEE.
- [4] D. Roest, A. Mesbah, and A. van Deursen. "Regression testing ajax applications: Coping with dynamism," Third International Conference on Software Testing, Verification and Validation, 2010, pp. 127-136. IEEE.
- [5] C. P. Bezemer, A. Mesbah, and A. van Deursen. "Automated security testing of web widget interactions," Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2009, pp. 81-90. ACM.
- [6] F. Ferrucci, F. Sarro, D. Ronca, and S. Abrahao. "A Crawljax Based Approach to Exploit Traditional Accessibility Evaluation Tools for AJAX Applications," Information Technology and Innovation Trends in Organizations, 2011, pp. 255-262. Springer.
- [7] C. Duda, G. Frey, D. Kossmann, and C. Zhou. "Ajaxsearch: crawling, indexing and searching web 2.0 applications," Proceedings of the VLDB Endowment 1.2, 2008, pp. 1440-1443. ACM.
- [8] comScore. "The U.S. Mobile App Report," comScore Whitepaper.
- [9] StatCounter Web Usage Stats, Jan 2012 - Jan 2015. <http://gs.statcounter.com/#all-comparison-ww-monthly-201201-201501>. 12 May 2015.
- [10] A. Nederlof, A. Mesbah, and A. van Deursen. "Software engineering for the web: the state of the practice," Companion Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 4-13. ACM.
- [11] S. Thummalapenta, K. V. Lakshmi, S. Sinha, N. Sinha, and S. Chandra. "Guided test generation for web applications," 35th International Conference on Software Engineering, 2013, pp. 162-171. IEEE.
- [12] A. M. Fard, and A. Mesbah. "Feedback-directed exploration of web applications to derive test models," ISSRE, 2013, pp. 278-287.
- [13] A. Marchetto, P. Tonella, and F. Ricca. "State-based testing of ajax web applications," 1st International Conference on Software Testing, Verification, and Validation, 2008, pp. 121-130. IEEE.
- [14] Mitmproxy. <https://mitmproxy.org/>. 12 May 2015.
- [15] Y. Maezawa, H. Washizaki, and S. Honiden. "Extracting interaction-based stateful behavior in rich internet applications," 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 423-428. IEEE.
- [16] H. Takagi, S. Saito, K. Fukuda, and C. Asakawa. "Analysis of navigability of Web applications for improving blind usability." ACM Transactions on Computer-Human Interaction v. 14, no. 3, 2007, article number 13. ACM.
- [17] W3 JavaScript APIs. http://www.w3.org/TR/#tr_Javascript_APIS/. 12 May 2015.
- [18] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. "A symbolic execution framework for javascript," IEEE Symposium on Security and Privacy, 2010, pp. 513-528. IEEE.
- [19] C. S. Jensen, M. R. Prasad, and A. Moller. "Automated testing with targeted event sequence generation," Proceedings of the 2013 International Symposium on Software Testing and Analysis, 2013, pp. 67-77. ACM.
- [20] C. Hu, and I. Neamtii. "Automating GUI testing for Android applications," Proceedings of the 6th International Workshop on Automation of Software Test, 2011, pp. 77-83. ACM.
- [21] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. de Carmine, and A. M. Memon. "Using GUI ripping for automated testing of Android applications," Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 258-261. ACM.
- [22] K. Nishiura, Y. Maezawa, H. Washizaki, S. Honiden, "Mutation Analysis for JavaScript Web Applications Testing," Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering, 2013, pp.159-165.
- [23] Y. Maezawa, H. Washizaki, Y. Tanabe and S. Honiden, "Automated Verification of Pattern-based Interaction Invariants in Ajax Applications," IEEE/ACM 28th International Conference on Automated Software Engineering, 2013, pp. 158-168. IEEE.
- [24] Y. Maezawa, K. Nishiura, H. Washizaki, S. Honiden, "Validating Ajax Applications Using a Delay-Based Mutation Technique," Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 491-502. ACM.

An Oracle based on Image Comparison for Regression Testing of Web Applications

Akihiro Hori*, Shingo Takada*, Haruto Tanno[†] and Morihide Oinuma[†]

*Dept. of Information and Computer Science, Keio University, Yokohama, Japan

{hori, michigan}@doi.ics.keio.ac.jp

[†]Software Innovation Center, NTT Corporation, Tokyo, Japan

{tanno.haruto, oinuma.m}@lab.ntt.co.jp

Abstract—Much work has been done on automating regression testing for Web applications, but most of them focus on test data generation or test execution. Little work has been done on automatically determining if a test passed or failed; testers would need to visually confirm the result which can be a tedious task. The difficulty is compounded by the fact that parts of a Web page (such as advertisements) may change each time the Web application is executed even though it has no bearing on the Web application function itself. We thus propose a test oracle for automatically determining the result of regression testing a Web application. The key point of our approach is the identification of parts that may change, which we call *variable region*. We first generate the expected result, by executing the original (pre-modification) Web application multiple times so that *variable regions* can be identified. Then, after the Web application is modified, regression testing is conducted by comparing the output of the modified Web application against the expected output. An evaluation confirmed the usefulness of our approach.

Keywords- web application testing; regression test; image comparison

I. INTRODUCTION

The Web application is a popular form of application due to the user basically only needing a Web browser to access it. They, however, tend to be modified frequently for various reasons, including bug fix, enhancements, and security attacks [1][2].

As with any software, each time a Web application is modified, it must go under regression testing, which checks that functions that performed correctly before modification still do. Although much work has been done on automatic regression testing [1][3][4][5][6][7][8], most focus on the automation of test data generation, selection, prioritization, or its execution.

An important part of regression testing is to check the result of test data execution against the expected result. This may be done automatically through means such as DOM tree comparison [8] and XML/HTML output [4]. However, in many cases, this is done manually [9][10], especially for testing that relies on visually checking the screen display (i.e., browser output) [11]. Such manual checking is time consuming as human testers must check each Web page one by one; this is compounded by the fact that checking each Web page needs to take place each time the application has been modified [10].

We thus focus on the automation of checking if the screen display has changed or not. Specifically, we target the development of a *test oracle* for Web applications.

A test oracle can be defined as having two essential parts [12]:

- 1) oracle information that represents expected output
- 2) an oracle procedure that compares the oracle information with the actual output

In our approach, we automatically generate the expected output for each test case by executing the test case multiple times and saving the resulting screen display as an image. After the Web application is modified, its screenshot is also saved and compared with the expected output. If the images are the same, the test case is said to have passed, otherwise it has failed.

Although the basic steps are simple, there is one important difference between the output of Web applications and conventional GUI applications. There may be regions within the output of Web applications that may change each time that Web page appears regardless of the actual results of the Web application execution. We call such a region as *variable region*. For example, many Web applications have pages that contain advertisements, which may change each time that page is shown. Such regions must be accounted for; otherwise even if the Web application result is correct, a change in the advertisement being shown will cause the image comparison to fail. Thus an important part of our approach is the elimination of these variable regions.

This paper thus proposes an oracle for the regression testing of Web applications. The main contributions are as follows:

- Automatic identification of variable regions by executing the Web application multiple times before modification.
- Automatic generation of an expected result (baseline image) for a given test case based on the identification of variable regions.
- Automatic “PASS/FAIL” judgement of test case execution by comparing screenshots taking into account variable regions.

The rest of this paper first starts with a discussion of related work. Section 3 then describes our approach. Section 4 evaluates our approach. Section 5 makes concluding remarks.

II. RELATED WORK

Web application testing focusing on the visual aspect of the output includes [11][13][14][15][16].

Stocco, et al. proposed an approach to migrate test suites based on DOM (Document Object Model) [17] to test suites based on images [11]. Although the goal of their work is different from ours, there are aspects similar to our work; specifically, the mapping of DOM elements with visual elements of the Web page.

Choudhary, et al. proposed an approach to detect cross-browser incompatibilities in Web applications [13]. Although the goal of their work is different from ours, they also target eliminating variable regions. However, their approach generates the expected output by executing the pre-modification Web application twice. If the variable regions *always* change, this would be sufficient, but this is not the case. Our approach solves this issue.

Selay, et al. [10] proposed an approach that compares images for regression test. Their approach was able to efficiently detect layout faults while neglecting insignificant variations. However, their “insignificant variation” does not include our variable region. Our approach thus differs from this perspective.

Applitools [15] is a tool that automatically tests Web applications in a variety of environments (e.g., OS, browsers), and saves screenshots. After the tests are executed, the saved images are shown one by one, and testers need to determine “PASS/FAIL” one by one. Our approach goes a step further as the “PASS/FAIL” determination is done automatically.

PhantomCSS [14] is a tool that automatically executes regression testing of Web applications by comparing screenshots of modified Web applications with screenshots of pre-modification Web applications. The goal of their tool is the same as ours, but variable regions can only be eliminated manually. Our approach solves this issue by automatically removing variable regions.

Screenster [16] is a tool that automatically determines “PASS/FAIL” of tests. Developers first record the initial result of executing a Web application. After making changes to the Web application, it is executed and the execution result is compared with the initial result. If the images are the same, the test is determined as pass. Otherwise, differences in the two results are highlighted. Testers then manually check if the differences can be ignored (in which case the test passes) or not (in which case the test fails). Although Screenster can automatically determine “PASS/FAIL”, it cannot handle the dynamic parts (variable region), while our approach can.

In sum, although much work has been done on visual testing of Web applications, the determination of “PASS/FAIL” is still done manually. PhantomCSS and Screenster can make an initial determination of “PASS/FAIL”, but they cannot automatically handle variable regions. Our proposed approach addresses this issue.

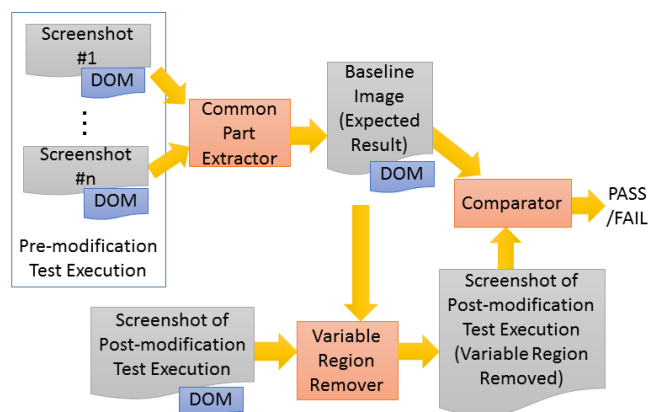


Fig. 1. Overview of proposed approach

III. IMAGE COMPARISON BASED TEST ORACLE

A. Overview of our approach

Fig. 1 shows the overview of our proposed approach. The basic steps are as follows:

- 1) Execute a test case multiple times before the program is modified, and produce n screenshots of the resulting Web page. We call this *pre-modification test execution*. The number of times (n times) is set by the tester.
- 2) Extract the common parts of n screenshots (subsection III-B). This eliminates the variable regions, and the resulting image serves as the expected result for the regression test.
- 3) Execute the test with the modified program, and produce a screenshot of the resulting Web page.
- 4) Remove the variable region from the resulting Web page (subsection III-C).
- 5) Compare the baseline image with the post-modification image (subsection III-D). If the two images match, then the regression test result is PASS. Otherwise, it is a FAIL.

B. Common Part Extraction

The common part extraction first divides each screenshot into several regions based on the DOM tree (Fig. 2). The DOM tree is traversed from its root until an element reaches a specified size, at which point all children of that element is deleted (Fig. 3). Each leaf in the remaining tree is considered as a “region”. Note that all n screenshots will have the same DOM tree structure, and thus the same resulting regions.

The element size is set by the tester. If the threshold takes too large a value, a region that contains both variable parts and non-variable parts will be designated as being a variable region. On the other hand, too small a value will lead to many regions which will cause unnecessary computation.

The common part extraction continues by comparing the regions one by one based on the position within the DOM tree. The set of regions that are the same with all n screenshots are

Screenshot #1 after pre-modification test



Fig. 2. Separation into region images

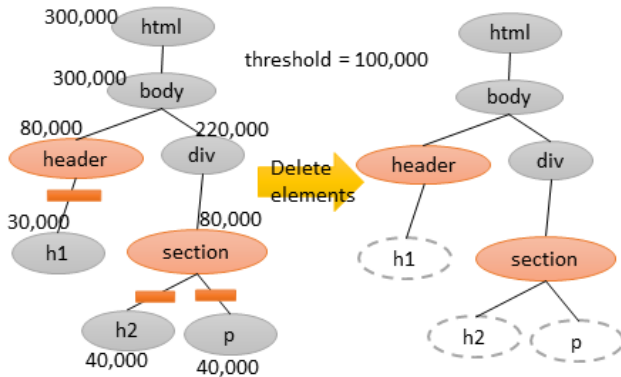


Fig. 3. The search and delete of elements in the DOM tree (Values express the size of the element)

stored in a log file (which we call *pre-modification common part log file*). Each log in this file is a pair of the image of the region and its DOM tree absolute path. This set of regions forms the baseline image, i.e., the expected output. Regions that differ at least once are considered as variable regions.

The comparison between each region, i.e., the comparison between two images, is done by calculating the distance between the two regions, and then checking if the distance is less than a specified threshold. If the distance is less than the threshold, the two regions (i.e., images) are considered to be the same. If it is greater than or equal to the threshold, they are considered to be different.

We adopted the χ^2 histogram distance for the region comparison, which is often used in image processing. It has been found to be accurate and computationally fast [13]. χ^2 histogram distance is computed as follows:

$$\chi^2(H_1, H_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)} \quad (1)$$

H_1 and H_2 are the histograms of the images. These histograms are the distribution of the luminance, i.e., i indicates the luminance value and $H(i)$ indicates the number of pixels that has luminance value i .

C. Variable Region Removal

The variable region removal process starts by first dividing the screenshot image of the post-modification Web page into regions in the same way as the common part extraction. The DOM tree is then used to remove the regions corresponding to the variable regions. Specifically, the DOM tree absolute path for each region is checked to see if it exists in the *pre-modification common part log file*. If it exists, then the image of that region will be compared with the corresponding baseline image in the next step. Otherwise, that region is considered as a variable region and ignored in the next step.

D. Comparison

This module compares the corresponding regions based on the DOM tree between the baseline image and the post-modification image with the variable regions removed. The algorithm used to compare each region is the same as the one used during the common part extraction (section III-B). If all regions are the same, the regression test has passed. Otherwise, it has failed.

E. Tool Implementation

We implemented our proposed approach on top of a testing tool that we had previously developed [18] [19]. In our previous tool, users first define what we call a *base scenario*, which corresponds to a set of steps that end-users take when they use a given Web page “normally”. Test cases are then generated by using the steps in the base scenario, and searching a knowledge-base to find similar steps. Found steps will include information such as constraints on user input and previously used test data that are used for the generated test cases.

The generated test cases are then executed using Selenium Web Driver [20], and the screen display is saved. Our previous work was able to automate many steps that are required for testing Web applications, but it could not automatically judge if a test case execution passed or failed. Our proposed approach makes this possible.

We list below values that the tester can specify:

- REPEAT indicates the value of n , i.e., the number of times the test is executed pre-modification. The default value is 3.
- SIZE is the threshold specifying the minimum size of elements in the DOM tree (Fig. 3). The default value is 100,000 pixels.
- DISTANCE is the threshold for determining if two regions are considered to be the same or not. The default value is 10.
- INTERVAL is the wait time between pre-modification tests. This is to account for cases where variable regions may change only after a certain amount of time has passed. The default value is 0.

The default values were specified based on preliminary experiments that we conducted.

IV. EVALUATION

We conducted a case study to evaluate our approach, focusing on the following three research questions:

- RQ1: Does our tool judge as correctly as humans?
- RQ2: Does our tool judge more quickly than humans?
- RQ3: Does our tool remove more variable regions when the number of times of the pre-modification test is bigger?

RQ1 checks how accurate our tool is compared to manual human comparison. RQ2 checks the time taken to compare images. Since automation is supposed to help humans, our tool needs to be as accurate (or nearly as accurate) as humans, and it also needs to be faster. RQ3 checks whether the repeated execution of pre-modification test is meaningful.

A. Target Applications

We targeted three demo Web applications: Zen Cart [21], Takai [22], and Welcart [23]. Zen Cart is a shopping cart software. Welcart is also a shopping cart software, but it is a WordPress plugin. Takai is a free Joomla template, that can be used to build Web sites. We chose these Web applications because they are open source, have variable regions, and have sample data available.

We used the provided programs as the original (pre-modification) applications. The modifications were done by changing the programs using the concept of mutation analysis [24], which basically changes or *mutates* a part of the code. We employed 26 basic mutation operators each of which changes one operator, such as changing $<$ to $>$. Although mutating the program basically means that we are injecting a bug into the program, the goal of this case study is to check if our tool can accurately judge if there has been a change in the output Web page, and *not* to evaluate the test suite. In a true testing

TABLE I
ACCURACY RATE (%)

	Tool	Human
Zen Cart	99.4	99.8
Takai	98.0	90.7
Welcart	100	100
Total	99.4	98.8

environment, there will be additional test cases that check for changes that were made to the program. Thus, some mutated code will result in the output Web page being different from the original output, while others will be the same. Adding new test cases to account for changes made to the program is out of scope of this paper.

Test cases were generated using our previously developed tool [18][19]. Twenty-eight test cases were generated for Zen Cart, five test cases for Takai, and twelve test cases for Welcart.

The case study was conducted using FireFox and specified the following values:

- REPEAT: 6 for RQ1 and RQ2; 1 to 6 for RQ3
- SIZE: 100,000
- DISTANCE: 15
- INTERVAL: 0

B. Manual Judgement

We asked five students to compare pairs of Web page images, and judge if each pair is the same or not. For each pair, one Web page image was the result of executing the original program, while the other was randomly chosen from the mutated results. The variable region was specified for each pair by highlighting the appropriate part of the Web page. In other words, each student subject needed to check the non-variable regions, and did not need to consider the variable region(s) when judging if a pair was the same or not. This is because in an actual testing environment, the tester will know in advance which part(s) of the Web page will change each time it is loaded.

C. Results

1) *RQ1: Accuracy*: We carefully analyzed both the results of our tool and the results of the humans manually, and checked if the judgement was correct or not.

Table I shows the results. The accuracy rate of our tool was nearly as high as the human result.

Tables II - V show the breakdown of the results. "Tool" denotes the result from our tool, while "Solution" denotes the correct result, which was obtained from careful manual analysis by the authors. "P" means *pass*, and "F" means *fail*. The numbers indicate the number of test data. So, for example, in Table II, our tool correctly identified 607 *passes* and 228 *fails*, while incorrectly identifying 5 *fails* as *passes*. The only cases of a *pass* incorrectly being identified as a *fail* occurred in the Takai Web application.

TABLE II
ZEN CART

		Solution	
		P	F
Tool	P	607	5
	F	0	228

TABLE III
TAKAI

		Solution	
		P	F
Tool	P	106	0
	F	3	41

TABLE IV
WELCART

		Solution	
		P	F
Tool	P	111	0
	F	0	249

TABLE V
TOTAL

		Solution	
		P	F
Tool	P	824	5
	F	3	518

TABLE VI
EXECUTION TIME (SECONDS)

	Tool	Human
Zen Cart	4.2	48.1
Takai	3.7	64.2
Welcart	2.4	27.7
Total	3.7	44.4

2) *RQ2: Execution Time*: We measured the execution time for our tool. Specifically, we measured the time taken for variable region removal and comparison (Fig. 1) We did not include the time taken for common part extraction because, if necessary, this can be executed when time is available such as at night.

The time for the five human subjects were also taken. Since they could judge *failed* comparisons more quickly than *passed* comparisons, we took the percentage of *passes* and *fails* into consideration and calculated a weighted average.

Table VI shows the results. This is the average time to compare one regression test execution. Our tool took less than one-tenth the time of the human subjects.

3) *RQ3: Repeat Count*: We executed our tool while changing the value of n from 1 to 6. Table VII shows the results. The value indicates the percentage of variable regions that our tool correctly identified. As expected, when the pre-modification test is executed more (i.e., the value of n is greater), the percentage of correct identification becomes higher.

D. Discussion

1) *RQ1: Accuracy*: Our approach had an accuracy of over 99%, which is comparable to human beings. The issues our approach had were as follows:

- Warning dialogs
Five out of the eight incorrect judgements were due to

not being able to capture the image of a warning dialog. For example, in Zen Cart, when there is an issue with registering user information, a warning dialog pops up. Our tool uses Selenium to capture images, but these pop up dialogs cannot be captured by Selenium. Thus our current implementation takes an ad hoc approach of using the “Print Screen” function of Windows. Unfortunately, when we were conducting pre-modification test, this method of capturing the dialog failed once. This led to the warning image to be handled as a variable region. In other words, it was not considered as a region image to be checked, and thus led that to be PASS instead of FAIL.

- Change in image size
Two out of the eight incorrect judgements were due to the image size becoming smaller by two pixels. Whenever there is a difference in image size, our tool first trims the larger image so that the images are of the same size. But our tool did not correctly trim an image.
- Timing of capturing a screenshot
The final incorrect judgement was due to the timing of capturing a screenshot by Selenium. Our tool currently waits three seconds after the Web page is loaded. The image of the Web page is then captured. This would handle cases where after the Web page is loaded, a widget is briefly shown, and then disappears. Unfortunately, in one case, a widget did not disappear after three seconds, and thus was captured.

Note that none of the above issues were due to our usage of χ^2 histogram distance for the region comparison. Since this approach only considers the distribution of the colors that are used in the Web page, technically there is a possibility that two completely different looking Web pages will have a χ^2 histogram distance of zero, i.e., those two pages are computed to be the same. This did not occur in our evaluation. Of course, more experiments are necessary to correctly conclude that χ^2 histogram distance is sufficient for our approach.

Furthermore, note that for Takai, the accuracy for manual judgement was much lower than the other two applications. This was due to a very small change in the Web page that the subjects were not able to detect. Specifically, in one case, the color of some text in a very small region changed from gray to black. Our tool was able to detect such cases, as the χ^2 histogram distance focuses on color.

2) *RQ2: Execution time*: As expected, our tool was faster than manual checking. For Zen Cart and Welcart, our tool was 11.5 times faster, while it was 17.4 times faster for Takai. The reason that our tool especially performed better for Takai was probably because Takai’s Web page was more complex than the other two, and thus it took subjects more time.

3) *RQ3: Repeat count*: The result of RQ3 found that the percentage of variable regions that are correctly identified is higher when the value of n (the number of times the pre-modification test is executed) is higher. We consider this from a probabilistic perspective.

Suppose that there are m variable regions in a Web applica-

TABLE VII
PERCENTAGE OF CORRECT IDENTIFICATION OF VARIABLE REGIONS

	n					
	1	2	3	4	5	6
Zen Cart	0	50.0	90.0	90.0	100	100
Takai	0	96.7	100	100	100	100
Welcart	0	76.7	96.7	100	100	100

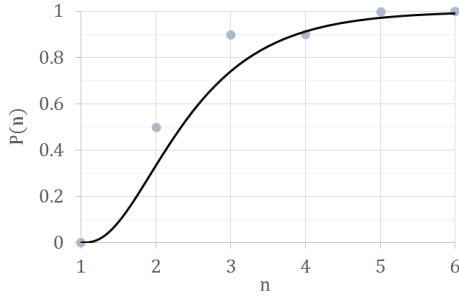


Fig. 4. Repeat time and Accuracy Rate

tion under test and that the pre-modification test was conducted n times. The probability $P(n)$ where the tool can correctly identify all m variable regions can be calculated as shown in equation (2).

$$P(n) = \prod_{k=1}^m \left\{ 1 - \left(\frac{1}{a_k} \right)^{n-1} \right\} \quad (2)$$

In equation (2), a_k means that the k th variable region has a_k contents that can be shown. In other words, for a given instance of a Web page, one of a_k possible contents is shown for the k th variable region. When pre-modification test is executed n times, the probability that the k th variable region is not removed (i.e., correctly identified) is the same as the same content being displayed consecutively n times. Assuming that the probability of one of the a_k contents being shown is the same, a content being displayed consecutively n times can be computed as $\left(\frac{1}{a_k} \right)^{n-1}$. Thus the probability of the k th variable region being correctly identified is a complementary event and can be computed as $1 - \left(\frac{1}{a_k} \right)^{n-1}$. Since the number of variable regions is m , we take the product of each variable region resulting in equation (2).

We consider the result of Zen Cart (Table VII), which had 6 variable regions. Of the 6 regions, 3 regions had an extremely high number of contents (i.e., the number of a_k was very large), and thus they can be ignored in terms of equation (2). As for the other 3 regions, the number of contents were 3, 3, and 4. Thus, if we assign $m = 3$, $a_1 = 3$, $a_2 = 3$, $a_3 = 4$ to equation (2), we obtain the curve in Fig. 4. This figure also contains the plots from Table VII.

E. Threats to Validity

The first threat to validity would be the use of students as the human subjects. Since the task was to compare Web page images, which anyone can do, we do not believe that this is a threat in terms of students vs professional. However, there is always the possibility that some humans are better at comparing than others, and we cannot discount this possibility as a threat.

The second threat to validity is the three Web applications we used. We chose the three because they were open source, have variable regions, and have sample data. Evaluation with

other Web applications should be conducted to strengthen our findings.

The third threat to validity are the test cases that were used. Test cases were generated using a tool that we had previously developed. We cannot completely deny this as a threat, since other test cases may result in Web pages that are difficult to compare. However, we have manually inspected the possible Web pages, and at least for the three Web applications, we do not believe that the generated test cases would be a threat.

The fourth threat to validity is the values we specified for the parameters REPEAT, SIZE, DISTANCE, and INTERVAL. Table VII showed that specifying the value of REPEAT as 6 had no effect on the result as the correct identification was 100% in all three Web applications. If anything, the value of REPEAT could have been set lower. The value for INTERVAL also had no effect as the three Web applications did not have any variable regions based on the amount of time needing to pass. The values for the other two parameters SIZE and DISTANCE were determined based on preliminary investigation, and these two may pose as threats. In fact, DISTANCE especially seems to be important. This is because if we used the default DISTANCE value of 10 (rather than the 15 we used in the evaluation), differences of 1 pixel would occur when dividing screenshots into regions. Although this seems to be a very small difference, this would lead to incorrectly identifying same images as different. In other words, regions that should be considered as the same would be considered as variable regions. Further evaluation should be done to investigate this.

V. CONCLUSION

We proposed an oracle for regression testing of Web applications. The oracle consisted of two parts: the expected result and the comparator. The key part of our oracle is accounting for variable regions, i.e., regions within the output Web page that may change each time it appears. The expected result is generated by executing the test multiple times so that variable regions can be identified and removed. The comparator checks the post-modification test execution result against the expected result.

An evaluation of our approach showed that our tool can identify the variable regions as accurately as, but quicker than, human subjects. We also showed that by repetitive execution of the pre-modification Web application improves the accuracy of the comparison.

As for future work, first we need to consider the optimization of thresholds. Our tool uses two thresholds: the minimum size of elements in the DOM tree to specify the region in each screenshot, and the χ^2 histogram distance threshold to determine if two regions should be considered as the same or not. The current (default) values are based on experience from applying our approach to several Web pages. But this may not always be optimal for other Web applications.

Second, although our approach had a very high accuracy, it was still not 100%. One very important issue that we need

to handle to raise the accuracy is to be able to handle dialogs that pop up as the result of executing a test case.

Finally, shortening the execution time further is another important part of future work.

REFERENCES

- [1] A. Marback, H. Do, and N. Ehresmann, "An effective regression testing approach for PHP Web applications," in *Proc. IEEE 5th International Conference on Software Testing, Verification, and Validation (ICST 2012)*, 2012, pp. 221–230.
- [2] S. N. A. Kamalzaman, S. M. Syed-Mohamad, S. Sulaiman, and K. Zamli, "Supporting maintenance of web applications using user-centered technique," in *Proc. 19th Asia-Pacific Software Engineering Conference*, 2012, pp. 43–49.
- [3] L. Xu, B. Xu, Z. Chen, J. Jiang, and H. Chen, "Regression testing for web applications based on slicing," in *Proc. 27th Annual International Computer Software and Applications Conference (COMPSAC 2003)*, 2003, pp. 652–656.
- [4] K. Dobolyi and W. Weimer, "Harnessing web-based application similarities to aid in regression testing," in *Proc. IEEE 20th International Symposium on Software Reliability Engineering (ISSRE2009)*, 2009, pp. 71–80.
- [5] M. Hirzel, "Selective regression testing for web applications created with Google Web Toolkit," in *Proc. 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools (PPPJ '14)*, 2014, pp. 110–121.
- [6] D. Garg, A. Datta, and T. French, "A two-level prioritization approach for regression testing of web applications," in *Proc. 19th Asia-Pacific Software Engineering Conference (APSEC 2012)*, 2012, pp. 150–153.
- [7] S. Mirshokraie and A. Mesbah, "JSART: JavaScript assertion-based regression testing," in *Proc. 12th International Conference on Web Engineering (ICWE 2012)*, 2012, pp. 238–252.
- [8] S. Raina and A. P. Agarwal, "An automated tool for regression testing in web applications," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–4, 2013.
- [9] V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "WebMate: A tool for testing web 2.0 applications," in *Proc. Workshop on JavaScript Tools (JSTools 12)*, 2012, pp. 11–15.
- [10] E. Selay, Z. Q. Zhou, and J. Zou, "Adaptive random testing for image comparison in regression web testing," in *Proc. 2014 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2014, pp. 1–7.
- [11] A. Stocco, M. Leotta, F. Ricca, and P. Tonella, "PESTO: A tool for migrating DOM-based to visual web tests," in *Proc. 14th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2014)*, 2014, pp. 65–70.
- [12] A. Memon, I. Banerjee, and A. Nagarajan, "What test oracle should I use for effective GUI testing," in *Proc. 18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, 2003, pp. 1–10.
- [13] S. Choudhary, H. Versee, and A. Orso, "A cross-browser web application testing tool," in *Proc. 26th IEEE International Conference on Software Maintenance (ICSM 2010)*, 2010, pp. 1–6.
- [14] "PhantomCSS," 2013. [Online]. Available: <https://github.com/Huddle/PhantomCSS>
- [15] "Applitoools," [Online]. Available: <https://applitoools.com/>
- [16] "Screenster," [Online]. Available: <http://www.creamtec.com/products/screenster/>
- [17] "W3C Document Object Model," [Online]. Available: <http://www.w3.org/DOM>
- [18] R. Lacanienta, S. Takada, H. Tanno, and M. Oinuma, "A knowledge-based approach for generating test scenarios for web applications," in *Proc. 25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013)*, 2013, pp. 166–171.
- [19] H. Saito, S. Takada, H. Tanno, and M. Oinuma, "Test data generation for web applications: A constraint and knowledge-based approach," in *Proc. 26th International Conference on Software Engineering and Knowledge Engineering (SEKE 2014)*, 2014, pp. 110–114.
- [20] "Selenium," [Online]. Available: <http://www.seleniumhq.org/>
- [21] Zen Ventures, LLC, "Zen Cart," [Online]. Available: <http://www.zen-cart.com/>
- [22] JoomlaWorks Ltd., "Takai," [Online]. Available: <http://www.joomlaworks.net/joomla-templates/free-templates/takai>
- [23] Collne Inc., "Welcart," [Online]. Available: <http://www.welcart.com/>
- [24] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

Towards the Anonymisation of RDF Data

Filip Radulovic
Ontology Engineering Group
ETSI Informáticos
Universidad Politécnica de Madrid
Madrid, Spain
fradulovic@fi.upm.es

Raúl García-Castro
Ontology Engineering Group
ETSI Informáticos
Universidad Politécnica de Madrid
Madrid, Spain
rgarcia@fi.upm.es

Asunción Gómez-Pérez
Ontology Engineering Group
ETSI Informáticos
Universidad Politécnica de Madrid
Madrid, Spain
asun@fi.upm.es

Abstract—Privacy protection in published data sets is of crucial importance, and anonymisation is one well-known technique for privacy protection that has been successfully used in practice. However, existing anonymisation frameworks have in mind specific data structures (i.e., tabular data) and, because of this, these frameworks are difficult to apply in the case of RDF data. This paper presents an RDF anonymisation framework that has been developed to address the particularities of the RDF specification. Such framework includes an anonymisation model for RDF data, a set of anonymisation operations for the implementation of such model, and a metric for measuring precision and distortion of anonymised RDF data. Furthermore, this paper presents a use case of the proposed RDF anonymisation framework.

I. INTRODUCTION

Large quantities of data are gathered and published every day by public and private companies and institutions. One key aspect of data publishing is the protection of the privacy of entities of interest (e.g., individuals), and failure to ensure the privacy can not only harm the reputation of a publisher, but can also compromise the privacy of entities of interest by making their private information available to third parties.

Ensuring the privacy of data while preserving data usefulness is not a simple task. Usually, removal of the data that explicitly identify the entity of interest, such as social security numbers or telephone numbers, does not alone ensure privacy since the remaining data can often be linked to other published data and used for identification purposes [1]. For example, Sweeney showed in an experiment that 87% of the U.S. population is likely to be uniquely identified based only on a combination of a ZIP code, gender, and date of birth [2].

Anonymisation is one technique for privacy protection that has been successfully applied in practice, and a number of anonymisation frameworks have been developed to this date. However, these frameworks are developed having in mind specific data structures, such as tabular data, and they are difficult to apply for the anonymisation of data that have different structures and formats. This is the case of RDF (Resource Description Framework) [3] data.

With the increasing amount of RDF data being published in the Web (usually as Linked Data), privacy issues are expected to emerge. Furthermore, privacy concerns hinder the publication of Linked Data in different sectors (e.g., healthcare, energy)

where the re-identification of individuals or other entities of interest can lead to social or legal issues.

This paper presents a framework for the anonymisation of RDF data. Such framework describes an anonymisation model for RDF data called *k-RDFanonymity*, as well as anonymisation operations for the implementation of the mentioned model and a metric for measuring the precision and distortion of anonymised RDF data. Furthermore, this paper also presents a use case of the presented RDF anonymisation framework.

The remainder of this paper is organised as follows. Section II describes related work, while Section III presents the framework for the anonymisation of RDF data. Section IV presents a use case of such framework and, finally, Section V draws some conclusions and includes ideas for future work.

II. RELATED WORK AND BACKGROUND

This section gives a brief description of related work. First, we describe the foundations of data anonymisation frameworks. Afterwards, we describe RDF and its particularities that are of interest for data anonymisation.

A. Data Anonymisation Frameworks

Anonymisation is a widely accepted and used framework for privacy-preserving data publishing that aims to ensure the privacy of data and balance data analysis and utility [4].

In privacy-preserving data publishing, there are several data attributes of the entity of interest that are taken into account:

- *Explicit identifiers* are attributes that explicitly identify the entity of interest (e.g., identifier of a person, property number of a building).
- *Quasi identifiers (QIDs)* are sets of attributes that can potentially identify the entity of interest. Usually, those are the attributes whose values can be found in other data sets and that can be then used for identification purposes (it is important to note that each attribute in a quasi identifier does not alone identify an entity of interest).
- *Sensitive attributes* are those attributes that describe some sensitive information about the entity of interest (e.g., salary, disease).
- *Non-sensitive attributes* are all attributes which do not belong to any of the previous categories.

Explicit identifiers, QIDs, and sensitive attributes can be considered to be private attributes of the entity of interest.

Non-sensitive attributes are not considered to be a privacy issue.

To illustrate these attributes, we present an example of a medical records with data about patients and their diseases (Table I). There are no definitive guidelines on how to properly classify attributes, so the classification can be a difficult task. The attributes present in our example data set can be classified as follows: *Id* is an explicit identifier since it can explicitly identify the patient that a record belongs to; a set of attributes *{Job, Age}* is a quasi identifier (QID) since it can be expected that the same set of attributes can appear in some other data set that can also contain additional data (but not diseases) that are sufficient for patient identification (e.g., name and surname); *Disease* is a sensitive attribute since it gives sensitive information about patients.

TABLE I: Example of patients' medical records data.

Id	Job	Age	Disease
1872	Teacher	24	HIV
1352	Lawyer	28	Flu
1453	Musician	32	Flu
1389	Writer	35	HIV
1463	Writer	36	HIV
1305	Lawyer	22	Flu
1435	Teacher	25	HIV
1058	Musician	38	Flu

Data anonymisation implies that explicit identifiers must be removed from the data set [4] and that the original QIDs are anonymised. Different anonymisation models have been developed having in mind tabular data structures (e.g., k-anonymity [5], l-diversity [6]), and these anonymisation models can be implemented by applying various anonymisation operations [4]:

- *Suppression* is a technique in which one or more values in a data set are removed or replaced with some special value, while removed or replaced values are not disclosed.
- *Generalisation* is a technique that transforms values into more general values, i.e., into new values that are less precise but still consistent with the original ones.
- *Anatomisation* implies that the relationship between the quasi identifiers and the sensitive values is removed, while the data is not modified. This is achieved by separating the data related to quasi identifiers from the data containing sensitive values and by providing the relationship between the two data sets by introducing an identifier.
- *Perturbation* is a technique in which the original data are replaced with noise or synthetic data in such a way that statistical analyses based on the perturbed data do not significantly differ from the statistical analysis of the original data [4]. Unlike previous techniques, perturbation does not preserve the truthfulness of the data and the perturbed data do not correspond to real world entities.

Anonymisation models and anonymisation operations are integral parts of a data anonymisation framework. By applying anonymisation operations, an anonymisation model is imple-

mented and thus, the privacy of entities of interest in a data set is ensured. Furthermore, a data anonymisation framework specifies various metrics for measuring the distortion and usefulness of anonymised data (e.g., precision and minimal distortion [7], [8]).

B. Anonymisation in Resource Description Framework

RDF is a specification for describing resources on the Web, where resources can be anything including documents, objects, people, or abstract concepts [3]. Unlike in tabular data formats (e.g., databases), where existing anonymisation frameworks have been successfully applied to this date, data in RDF are structured in a different manner as a graph.

The key concept in RDF is an *RDF statement* (*s,p,o*), also called an *RDF triple*, which consists of a subject, a predicate, and an object, and which encodes a claim about the world. Each RDF triple implies the existence of a relationship that holds between two resources denoted by a subject and an object. Relationships in RDF triples are denoted by a predicate, also called a property, and are directed from subject to object.

While in tabular data formats attribute values of entities of interest are stored in columns, attribute values of entities of interest described in RDF appear as resources, either as IRIs or as literals, that describe these entities of interest. In an RDF graph, literals appear only as objects, while IRIs can appear in several places: as subjects, as predicates, or as objects. Therefore, attribute values of entities of interest can have different forms and can appear in different places in the RDF descriptions of these entities of interest.

In an RDF graph, resources can be classified according to categories specified by classes that belong to a specific *vocabulary*. Vocabularies are used in combination with RDF for providing semantic information about resources, and are defined using a specific language (e.g., RDF Schema or OWL). Relationships between an RDF resource and its class are defined through an *rdf:type* property. These relationships also describe resources in RDF.

The particularities of the RDF model imply difficulties in the direct application of existing anonymisation frameworks, which were developed having in mind different data structures and formats. Therefore, in order to successfully anonymise RDF data, anonymisation frameworks that address the particularities of RDF are needed. Although some effort in this direction exists [9], it addresses only generalisation and suppression, and it does not include any anonymisation metrics.

III. A FRAMEWORK FOR THE ANONYMISATION OF RDF DATA

This section describes a framework for the anonymisation of RDF data. Such framework is based on existing anonymisation frameworks and has been specifically defined to take into account the particularities of the RDF specification. It consists of an anonymisation model, of a set of anonymisation operations, and of an anonymisation metric adapted to fit the RDF specification.

A. Privacy-related Entity Attributes in RDF

One characteristic of RDF, which is of relevance for the problem of anonymisation, is that entity attribute values can appear in different places and forms in the description of resources that represent entities of interest. Because of this, privacy-related attributes in RDF are more difficult to identify and addressing the privacy of RDF data can be a complex task.

In the RDF description of entities of interest, values of privacy-related entity attributes (explicit identifiers, quasi identifiers, and sensitive attributes) can appear in:

- *Resource IRI*. Values for all three types of attributes can appear in the IRI of a resource, exposing them to humans.
- *Datatype property value*. Values for all three types of attributes can appear as literals in datatype property values in a resource description (object in a statement).
- *Object property value*. Values for all three types of attributes can appear as IRIs in object property values in a resource description (object in a statement).
- *Property IRI*. Values for attributes that belong to quasi identifiers and for sensitive attributes can appear in the IRI of a property in a resource description. Having explicit identifiers appear as property IRIs is not expected.
- *Related resource*. Values for all three types of attributes can also appear in more complex scenarios in the description of resources that are related to resources that represent entities of interest through RDF properties. In this case, values can appear in all scenarios presented above: in resource IRI, datatype property value, object property value, property IRI, or another related resource.

Next, we present an example of the previous scenarios¹ (Listing 1) in which the first record from Table I is described in RDF using the Turtle syntax. In this example, the identifier appears in the resource IRI, the age appears as a datatype property value, the job appears as an object property value, and the disease appears as a property IRI.

```
1 <http://example.com/resource/Person/1872> a foaf:Person;
2   foaf:age "24"; ex:hasJob ex:Teacher; ex:hashIV "true".
```

Listing 1: Example of a patient's medical record in RDF.

B. RDF Anonymisation Model

This section presents an anonymisation model for RDF, called *k-RDFAnonymity*. This model is inspired by the *k*-anonymity model [1], [5] developed by Samarati and Sweeney for the anonymisation of tabular data.

Definition 1. A subgraph G_r of an RDF graph G ($G_r \in G$) describes an entity of interest represented by a resource r if G_r is the union of all the subgraphs of G that include information about attributes that describe the entity of interest represented by r , regardless of whether r is the subject or the object of statements in these subgraphs.

Definition 2 (Equivalence). Graphs G_{r_1} and G_{r_2} are equivalent ($G_{r_1} \equiv G_{r_2}$) $\Leftrightarrow \forall (s \neq r_1, p, o \neq r_1) \in G_{r_1} \exists (s \neq r_2, p, o \neq r_2) \in G_{r_2} \wedge \forall (r_1, p, o) \in G_{r_1} \exists (r_2, p, o) \in G_{r_2} \wedge \forall (s, p, r_1) \in G_{r_1} \exists (s, p, r_2) \in G_{r_2}$.

¹In the sake of simplicity, we omit prefix and datatype declarations in all the examples.

$\in G_{r_2} \wedge \forall (r_1, p, o) \in G_{r_1} \exists (r_2, p, o) \in G_{r_2} \wedge \forall (s, p, r_1) \in G_{r_1} \exists (s, p, r_2) \in G_{r_2}$.

Definition 3. In an RDF graph G , with $QID(G)$ we denote a set of QID attributes that describe any entity of interest represented by a resource in G .

Definition 4. A subgraph G_r of an RDF graph G describes an entity of interest represented by a resource r with respect to $QID(G)$, written $G_r(QID(G))$, if G_r includes information about all QID attributes.

Definition 5 (k-RDFAnonymity). Let I be a set of resources that represent entities of interest described in an RDF graph G , and let $QID(G)$ be a set of QID attributes that describe these entities of interest. *k*RDF-anonymity in G is satisfied $\Leftrightarrow \forall G_r(QID(G)) \in G, r \in I, \exists r_s \in I, s \in [1, k-1] \Rightarrow \forall G_{r_s}(QID(G)) \in G, G_{r_s} \equiv G_r$. An RDF graph that satisfies this premise is called *k*-RDFAnonymous.

In a *k*-RDFAnonymous graph, each resource r that represents an entity of interest cannot be distinguished from *k*-1 other resources that represent entities of interest in a graph with respect to $QID(G)$. Therefore, the probability of identifying a specific resource based on resource descriptions with respect to $QID(G)$ is $1/k$.

C. RDF Anonymisation Operations

This section presents different anonymisation operations that can be used for implementing the *k*-RDFAnonymity model. These operations are based on the anonymisation operations presented in Section II-A and address private attributes, i.e., explicit identifiers, quasi identifiers, and sensitive attributes.

1) *Generalisation and Suppression*: In the context of RDF, suppression denotes that a resource (i.e., an IRI or literal) is completely removed or replaced with some specific resource while the original replaced resource is not disclosed in any way. Generalisation denotes that the original resource is replaced with other resource that describes a more general concept.

The starting point of generalisation and suppression is a domain generalisation hierarchy of an attribute, in which the elements of the hierarchy are resources that represent attribute values (i.e., that include information about that attribute). Figure 1 shows generalisation hierarchies for the *Age* attribute (literals), the *Job* attribute (IRIs representing a class from a specific vocabulary), and the *Disease* attribute (properties).

While generalisation implies the use of more general resources from a hierarchy (e.g., *ex:Job* instead of *ex:Musician*), suppression implies the complete removal of a resource or the use of the resource at the top of the hierarchy (e.g., *owl:Thing* instead of *ex:Musician*).

Since resources that include information about explicit identifiers unequivocally identify entities of interest, these resources have to be suppressed, while resources that include information about QID and sensitive attributes can be generalised or suppressed, depending on the concrete scenario. Next, we describe generalisation and suppression through different scenarios depending on the position in which resources that include information about explicit identifiers, QID and sensitive

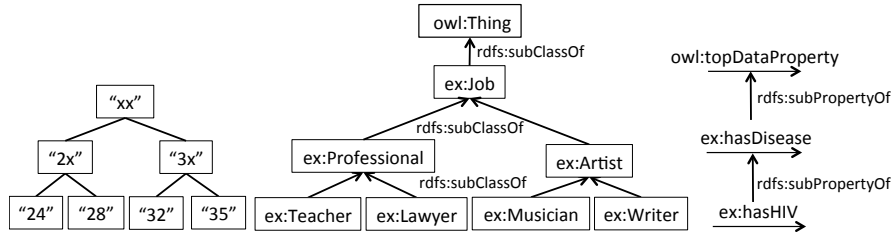


Fig. 1: Generalisation hierarchies for Age (left), Job (center) and Disease (right).

attributes appear in the description of a resource that represents an entity of interest (Section III-A):

- *Resource IRI*. If a resource that includes information about an explicit identifier appears in the IRI of the resource that represents an entity of interest, it cannot be simply removed from the graph because it would result in the loss of connections between nodes. Therefore, it has to be suppressed by replacing the original IRI with another one. This can be achieved through: i) simple replacement of the IRI with some arbitrary IRI. In this case, the uniqueness of each IRI has to be ensured; ii) encoding of the IRI by using encryption or hashing. In this case, the encryption or hashing function should ensure that the original IRI cannot be easily recovered, and that new IRI is unique; or iii) replacing the IRI node with a blank node. In this case, the original resource that includes information about the explicit identifier is suppressed and the original IRI is not disclosed.

Resources that include information about QID and sensitive attributes that appear in the IRIs of resources that represent entities of interest can be addressed by generalising or suppressing the original IRIs, using the previously defined domain generalisation hierarchies.

It is important to note that generalisation and suppression of IRIs breaks the uniqueness of the IRIs in a graph. This can be solved by introducing additional unique identifier values as part of the IRI of each resource.

- *Property values*. Resources that include information about explicit identifiers that appear either as datatype or as object property values are addressed by complete suppression, i.e., by removing property values from the graph. Resources that include information about QID and sensitive attributes that appear either as datatype or as object property values can be addressed by generalising or suppressing the original resources that appear as property values. In some cases, this operation has to be supported by the vocabulary design.
- *Property IRI*. Resources that include information about QID and sensitive attributes that appear as property IRIs can be addressed by generalisation through the use of super-properties.

Listing 2 presents an example of applying generalisation and suppression on the RDF data presented in Listing 1. The resource that includes information about the explicit identifier is

addressed by suppressing the original IRI with another one. Age (which is described as a datatype property value and includes information about a QID attribute), job (which is described as an object property value and also includes information about a QID attribute), and disease (which is described as a property IRI and includes information about a sensitive attribute) are addressed by generalising by one level in the generalisation hierarchy.

```

1 <http://example.com/resource/Person/Per01> a foaf:Person;
2   foaf:age "2x"; ex:hasJob ex:Professional;
3   ex:hasDisease "true".

```

Listing 2: Example of generalisation and suppression in RDF.

Since generalisation reduces the semantic precision of information and since for an RDF graph G there can be multiple generalisations, different generalisations of an RDF graph G are characterised by different semantic precisions of information.

Definition 7 (Graph generalisation). An RDF graph G_g is a generalisation of an RDF graph G with respect to $QID(G)$, written $G_g(QID(G)) \geq G(QID(G))$, if some or all resources that include information about QID attributes in a graph G are generalised.

Definition 8 (kRDF-minimal generalisation). Let an RDF graph G_g be a generalisation of an RDF graph G with respect to $QID(G)$. $G_g(QID(G))$ is said to be the k-RDFminimal generalisation of an RDF graph G with respect to $QID(G) \Leftrightarrow G_g(QID(G))$ is a k-RDFanonymous graph $\wedge \forall G_i(QID(G)): G_i(QID(G)) \geq G(QID(G)), G_g(QID(G)) \geq G_i(QID(G)), G_i(QID(G))$ is a k-RDFanonymous graph $\Rightarrow G_g(QID(G)) \equiv G_i(QID(G))$.

2) *Anatomisation*: In the context of RDF, anatomisation implies that resources that include information about sensitive attributes are not directly connected to resources r_i that represent entities of interest. Instead, resources that include information about sensitive attributes are grouped into several groups which describe how many resources r_i belong to each group. All resources r_i are then connected to these groups. This way, for each resource r_i it is known to which group it belongs to and, hence, it is only known with how many other resources that represent entities of interest r_i shares the sensitive information from a particular group.

Listing 3 shows an example of applying anatomisation on the RDF data presented in Listing 1.

From the previously described graph, for any disease group there is only information about how many patients have each

disease. Therefore, for any resource (patient) it is not explicitly known which disease is associated with it.

```

1 <http://example.com/resource/Person/Per01> a foaf:Person;
2   foaf:age "24"; ex:hasJob ex:Teacher;
3   ex:inGroup <http://example.com/resource/Group/01>.
4 <http://example.com/resource/Group/01> a ex:DiseaseGroup;
5   ex:hasDisease <http://example.com/resource/Disease/X>;
6   ex:hasDisease <http://example.com/resource/Disease/Y>.
7 <http://example.com/resource/Disease/X> a ex:Disease;
8   ex:name "HIV"; ex:cardinality "4".
9 <http://example.com/resource/Disease/Y> a ex:Disease;
10  ex:name "Flu"; ex:cardinality "4".

```

Listing 3: Example of anatomisation in RDF.

3) *Perturbation*: In the context of RDF, perturbation is an operation which replaces original resources in such a way that the semantics of resources affected is also changed, while preserving the statistical information of the original RDF graph. This can be achieved by: i) adding noise to the RDF data in such a way that the semantics of data is changed; ii) swapping resources by assigning to a resource that represents an entity of interest the description of some other resource that represents another entity of interest; or iii) generating synthetic resources.

Listing 4 presents an example of applying perturbation on the RDF data presented in Listing 1. In this example, noise has been introduced to the resource describing age, while resources describing job and disease have been swapped.

```

1 <http://example.com/resource/Person/Per01> a foaf:Person;
2   foaf:age "22"; ex:hasJob ex:Musician; ex:hasFlu "true".

```

Listing 4: Example of perturbation in RDF.

D. RDF Information Metrics

In the situation where for a given RDF graph there exist multiple k-RDFanonymous graphs, the decision on which k-RDFanonymous graph is the best to use for privacy protection can be a difficult task. In order to provide information that can help in making this decision, we have defined *RDFprec*, a precision metric of a k-RDFanonymous graph. This metric is based on the precision metric for tabular data developed by Sweeney [8] and can be used for defining the minimal distortion of an RDF graph.

Definition 9. With DGH_{a_i} we denote a domain generalisation hierarchy of an attribute a_i which describes an entity of interest represented by a resource. With $|DGH_{a_i}|$ we denote the number of levels in DGH_{a_i} , where the lowest level in a hierarchy is level 0. With v_{ij} we denote a resource which includes information about the value of an attribute a_i , and which describes an entity of interest represented by a resource r_j . With $h(v_{ij})$ we denote a height of a resource v_{ij} in DGH_{a_i} , where a resource v_{ij} at the lowest level in the hierarchy has height 0. With $|r|$ we denote the number of resources r that represent entities of interest in a graph G , i.e., those resources that have to be anonymised.

An example of a domain generalisation hierarchy is shown on Figure 1. In this example, in the case of a resource that describes age there are 2 levels, while in the case of a resource that

describes jobs there are 3 levels in the domain generalisation hierarchy. Resource “2x” is on the first level in the hierarchy.

Definition 10 (RDFprec). Let G be an RDF graph, G_g be a generalisation of G , v_{ij} be a generalised resource from G_g which includes information about the value of an attribute a_i and which describes an entity of interest represented by a resource r_j , DGH_{a_i} be the domain generalisation hierarchy of an attribute a_i , n_a be the number of generalised attributes, and m_i be the number of entities of interest represented by resources r_j in which a resource that includes information about an attribute a_i is generalised. The precision of G_g , written $RDFprec(G_g)$ is defined with the following formula:

$$RDFprec(G_g) = 1 - \frac{\sum_{i=1}^{n_a} \sum_{j=1}^{m_i} \frac{h(v_{ij})}{|DGH_{a_i}|}}{|r| * n_a}$$

If all resources in an RDF graph G_g are generalised to the highest level in a domain generalisation hierarchy, each $h(v_{ij}) = |DGH_{a_i}|$ and $RDFprec(G_g) = 0$. Contrary, if there are no resources that are generalised, each $h(v_{ij}) = 0$ and the $RDFprec(G_g) = 1$.

As an example, precision of the example RDF graph G presented in Listing 2, which consists of only one resource that represents an entity of interest, with respect to the domain generalisation hierarchy on Figure 1 is

$$RDFprec(G) = 1 - \frac{1/2 + 1/3 + 1/2}{1 * 3} = \frac{5}{9}$$

Definition 11 - RDF minimal distortion. Let an RDF graph G_g be a generalisation of an RDF graph G with respect to $QID(G)$. $G_g(QID(G))$ is said to be the RDFminimal distortion of an RDF graph G with respect to $QID(G) \Leftrightarrow G_g(QID(G))$ is a k-RDFanonymous graph $\wedge \forall G_i(QID(G))$: $RDFprec(G) \geq RDFprec(G_i)$, $RDFprec(G_i) \geq RDFprec(G_g)$, $G_i(QID(G))$ is a k-RDFanonymous graph $\Rightarrow G_g(QID(G)) \equiv G_i(QID(G))$.

IV. USE OF THE RDF ANONYMISATION FRAMEWORK

This section presents a use case of the RDF anonymisation framework presented in this paper. In this use case, we examine different generalisations of an RDF graph G that describes the medical records presented in Table I, based on the domain generalisation hierarchies presented in Figure 1.

For graph G six different suppressions and five different generalisations are possible. In this example, we focus only on generalisations that are addressed through a generalisation of the QID attributes, which include *Job* and *Age*.

The original RDF graph G with no generalisation, written $G_{[0,0]}$, describes the values for the *Job* and *Age* attributes at the lowest (zero) level in the domain generalisation hierarchies, and corresponds to the data in Table I. In this case, the k constraint is 0 and $RDFprec(G_{[0,0]}) = 1$. Listing 5 shows the excerpt of the original graph $G_{[0,0]}$ which includes information about teachers and lawyers.

Possible generalisations of graph G are: $G_{[0,1]}$ (meaning *Job* is not generalised while *Age* is generalised by one level), $G_{[1,0]}$, $G_{[1,1]}$, $G_{[2,0]}$, and $G_{[2,1]}$. Graphs $G_{[0,1]}$, $G_{[1,1]}$, and $G_{[2,1]}$ are k-RDFanonymous for $k=2$, while $G_{[1,0]}$ and $G_{[2,0]}$ are not k-RDFanonymous since in these cases $k = 0$.

```

1 <http://example.com/resource/Person/Per01> a foaf:Person;
2   foaf:age "24"; ex:hasJob ex:Teacher; ex:hasHIV "true".
3 <http://example.com/resource/Person/Per02> a foaf:Person;
4   foaf:age "28"; ex:hasJob ex:Lawyer; ex:hasFlu "true".
5 <http://example.com/resource/Person/Per06> a foaf:Person;
6   foaf:age "22"; ex:hasJob ex:Lawyer; ex:hasFlu "true".
7 <http://example.com/resource/Person/Per07> a foaf:Person;
8   foaf:age "25"; ex:hasJob ex:Teacher; ex:hasHIV "true".

```

Listing 5: Excerpt of the original graph with no generalisation.

Listing 6 presents the excerpt of the generalisation graph $G_{[0,1]}$ related to the excerpt presented in Listing 5. We can observe that, for any given combination of resources that include information about the *Job* and *Age* attributes, there are two resources that represent patients that are described with that same combination. For example, *Per01* and *Per07* are both teachers that are between 20 and 30 years old, and they both have HIV. Similarly, *Per02* and *Per06* are both lawyers that are between 20 and 30 years old, and they both have flu.

```

1 <http://example.com/resource/Person/Per01> a foaf:Person;
2   foaf:age "2x"; ex:hasJob ex:Teacher; ex:hasHIV "true".
3 <http://example.com/resource/Person/Per02> a foaf:Person;
4   foaf:age "2x"; ex:hasJob ex:Lawyer; ex:hasFlu "true".
5 <http://example.com/resource/Person/Per06> a foaf:Person;
6   foaf:age "2x"; ex:hasJob ex:Lawyer; ex:hasFlu "true".
7 <http://example.com/resource/Person/Per07> a foaf:Person;
8   foaf:age "2x"; ex:hasJob ex:Teacher; ex:hasHIV "true".

```

Listing 6: Excerpt of a generalisation of the original graph – $G_{[0,1]}$.

Among the three 2-RDFanonymous graphs, $G_{[0,1]}$ generalises *Age* by one level, $G_{[1,1]}$ generalises both *Job* and *Age* by one level, and $G_{[2,1]}$ generalises *Job* by two levels and *Age* by one level. Furthermore, since $G_{[1,1]}$ is generalisation of $G_{[0,1]}$, and $G_{[2,1]}$ is a generalisation of both $G_{[1,1]}$ and $G_{[0,1]}$, $G_{[0,1]}$ is the *k-RDFminimal* generalisation of G .

Since there are three generalisation graphs that are 2-RDFanonymous, by calculating *RDFprec* for each graph it can be determined which of the three graphs has minimal distortion. In this use case, $\text{RDFprec}(G_{[0,1]}) = 0.75$, $\text{RDFprec}(G_{[1,1]}) = 0.58$, and $\text{RDFprec}(G_{[2,1]}) = 0.42$. Therefore, the *RDFminimal* distortion of a graph G that satisfies 2-RDFanonymity is $G_{[0,1]}$.

V. CONCLUSIONS AND FUTURE WORK

This paper has presented a framework for the anonymisation of RDF data that addresses the particularities of the RDF specification and can help in preserving the privacy of entities of interest in RDF data sets. Such framework describes an anonymisation model, several anonymisation operations, and an anonymisation metric.

The anonymisation model presented in this paper helps in preserving the privacy of RDF data sets. However, in order to ensure the maximum possible protection of privacy in RDF data by implementing such anonymisation model, it is first necessary to correctly identify resources that include information about QID and sensitive attributes, which is a difficult task. In those cases when these resources are not correctly identified, anonymisation can lead to an overprotection of the RDF data

(i.e., lowering the precision of the anonymised data) or to a failure in ensuring the privacy of the RDF data.

The anonymisation model presented in this paper, although it ensures the protection of privacy to a certain level, is vulnerable to different kinds of attacks, such as in those cases when the order of entities of interest can compromise their privacy, or when subsequent releases of the same private information take place. Therefore, future work includes additional formalisations and recommendations on the implementation of the model to address those potential issues. These formalisations can be based on future case studies that will investigate the strength of the anonymisation model.

The framework for the anonymisation of RDF data described in this paper presents an initial effort towards the privacy protection of RDF data. Therefore, one line of future work consists of enriching the anonymisation framework described in this paper with other anonymisation models, besides the one presented in this paper, which can be based on the already existing set of models developed for tabular data. Furthermore, future work consists of including into the RDF anonymisation framework additional anonymisation metrics, and development of anonymisation algorithms. A variety of anonymisation models, metrics, and algorithms can help in achieving better privacy protection and will provide a comprehensive anonymisation framework for the privacy protection of RDF data.

ACKNOWLEDGMENTS

This work is supported by the 4V project (TIN2013-46238-C4-2-R), funded by the Spanish Ministry of Economy and Competitiveness, and by the FPU grant (FPU2012/04084) of the Spanish Ministry of Education, Culture and Sport.

REFERENCES

- [1] L. Sweeney, "k-Anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, pp. 557–570, 2002.
- [2] L. Sweeney, "Uniqueness of simple demographics in the U.S." Carnegie Mellon University, School of Computer Science, Data Privacy Laboratory, Tech. Rep., 2000.
- [3] G. Klyne, J. J. Carroll, and B. McBride, "RDF 1.1 Concepts and Abstract Syntax. Available online: <http://www.w3.org/TR/rdf11-concepts/>. Last retrieved on 10.12.2014." World Wide Web Consortium, Tech. Rep., 2014.
- [4] B. C.M. Fung, K. Wang, A. Wai-Chee Fu, and P. S. Yu, *Introduction to Privacy-Preserving Data Publishing: Concepts and Techniques*. Chapman & Hall/CRC, 2010.
- [5] P. Samaratiy and L. Sweeney, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," SRI International, Tech. Rep., March. 1998.
- [6] M. Ashwin, J. Gehrke, D. Kifer, and M. Venkatasubramaniam, "l-diversity: Privacy beyond k-anonymity," in *Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE)*, Atlanta, GA, USA, pp. 24–35, 2006.
- [7] P. Samaratiy, "Protecting respondents' identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, pp. 1010–1027, 2001.
- [8] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, pp. 571–588, 2002.
- [9] A. Gkoulalas-Divanis, S. Kotoulas, L. Vanessa, and M. L. Sbodio, "Guaranteeing anonymity in Linked Data graphs," International Patent PCT/US2014/033 261, 2014.

An Information Retrieval Model using Query Expansion based on Ontologies in the Computer Science Domain

Bonnie G. Carranza Chávez*, Andrés Melgar*†

* Grupo de Reconocimiento de Patrones e Inteligencia Artificial Aplicada,
Pontificia Universidad Católica del Perú, Lima, Perú

† Sección de Ingeniería Informática, Departamento de Ingeniería,
Pontificia Universidad Católica del Perú, Lima, Perú

Abstract—This paper presents a model that aims to support knowledge retrieval stored in digital repositories through domain ontologies. In this model the ontology contains concepts and relationships which describe a specific part of the world. The model mechanisms aim to reduce the impact of some of the main obstacles identified in the Information Retrieval process such as user specific characteristics, natural language characteristics or retrieval systems limitations. As a result the user, by providing a query to the system, can retrieve relevant information which better meet his information need. A prototype was developed to demonstrate the feasibility of the model using queries in the computer science domain.

Index Terms—information retrieval, query expansion, ontology

I. INTRODUCTION

In the past years, we have been witness to an exponential growth of digital information [1] which was triggered by the continuous development of IT. From a user's point of view, the traditional way to meet his information need would be performing an exhaustive review of contents from physical and digital documents [2]. However, this process clearly demands a considerable investment of time and effort, for that reason this alternative may not be possible for many users.

In order to lighten this process, Information Retrieval Systems (IR systems) progressively emerged [3]. These systems represented a tool for, in an automated way, retrieving useful information corresponding to the user's query, which at the same time lightened to a certain degree the difficulties of the manual exhaustive traditional search process [4]. However, nowadays these systems not necessarily present an ideal behavior. In first place, since they are not always able to effectively interpret what users want and need, it is not unusual they provide irrelevant documents. In second place, due to intrinsic characteristics of Natural Language (NL) such as: words ambiguity, context dependencies, the fact that a word may have different domain-specific meanings and the fact that a concept may be expressed by different words. As a result, it is common that documents relevant for the user are omitted, or that excessive information that does not meet user requirements is delivered.

In this paper we propose an alternative to the retrieval information problem so that retrieved documents are to a certain degree more relevant. This retrieval is treated under the

Query Expansion (QE) approach for which knowledge models such as ontologies are used. In specific, we developed an ontology in the Computer Science (CS) domain in the scope of a university curricula and a prototype to test the model. After analyzing the results of tests, it was concluded the integration of components succeeded on retrieving information relevant for the user and overcoming to a degree some of the obstacles identified for the retrieval process.

II. LITERATURE REVIEW

A. Information Retrieval

IR can be understood as the scientific discipline in charge of the analysis, design and implementation of computer systems which deal with the representation, storage, organization and access to non-structured information, and can provide answers to user queries [5]. The retrieved information which from the user point of view meet the stated query is called "relevant". The IR discipline focus on the maximization of retrieved relevant documents while at the same time minimizing the retrieval of non-relevant documents. These objectives can be quantified through the use of precision (ratio of the number of relevant documents retrieved to the total number of documents retrieved) and recall (ratio of the relevant documents retrieved to the total number of relevant documents) metrics [6].

B. Query expansion

Usually, users tend to formulate short queries instead of carefully built ones. Such short queries lack of words that if were provided, could be very useful search terms [7]. The QE goal is to add new meaningful terms to the initial query [8]. For example, for a query stating *Pilas* which is an ambiguous plural-form Spanish word that may refer to batteries, cells, heaps and stacks, adding the word *Baterias* (batteries) to the query would be meaningful because it would help the system to identify the domain the user is trying to query about. This addition would represent a QE.

C. Ontologies in IR

A conceptualization is an abstract, simplified view of the world. An ontology is an explicit specification of a conceptualization [9]. It consists of entities, attributes, relationship,

and axioms in a human understandable and machine readable format [10]. In recent years, ontologies have been adopted in many business and scientific communities as a way to share, reuse and process domain knowledge. For example, an ontology in the animal diseases domain developed by a medical expert would represent a base of knowledge which could be used by software developers to create applications to diagnose an animal illness from the symptoms [11]. Ontologies are increasingly being used in IR research as knowledge to support semantic search [12], [13]. For an ontology based IR system, when the user inputs the QE, the system tries to insert the ontology knowledge to enhance the QE in order to increase the probability of relevancy [10]. In [14] authors discuss that it isn't optimal the using of general purpose ontologies like WordNet for specific domains because it could lead to the losing of precision. For that reason, they introduced a QE algorithm for medical IR using concepts from the MeSH ontology. A different approach was proposed in [10], where the author introduced ontologies into QE and made a deep use of semantic relations of concepts to expand query keywords and to make the retrieval results more accurate. In [15] authors used automated QE with the support of ontologies. The objective of their QE in data integration proposal is to extend the results of a given query in a semantically meaningful way. They focused in the integration of different sources, and over this unification performing the QE.

III. PROPOSED MODEL FOR INFORMATION RETRIEVAL

The proposed model (see figure 1) was designed to facilitate IR using both ontologies and user information as input for the QE. It consists of 5 layers with a total of 10 components:

- **Visualization Layer:** is used to get the input from user and to show the outputs.
- **Support Layer:** contains support components in charge of coordinating all interactions and preparing the query for the expansion.
- **Retrieval Layer:** retrieves the documents based on the expanded query and the tagged documents.
- **Expansion Layer:** in charge of coordinating the overall QE process. This component has two sub-components, the Equivalence Handler and the Ambiguity Handler.
- **Data Access Layer:** retrieves information from the ontology, the documents repository and the user information DB.

A. Preprocessing Handler

This component aims to reduce the difficulties determined by the difference between how is knowledge stated in documents and how it was formulated in the query. In order to reduce some of the blurring effects of NL characteristics over the retrieval effectiveness two mechanisms are proposed. The first one, the stopwords removal mechanism (SRM), will perform a removal of those words which do not make a significant contribution of relevant concepts. The second one, the lemmatization mechanism (LM), will support the simplification of both, the query and the documents tags, in

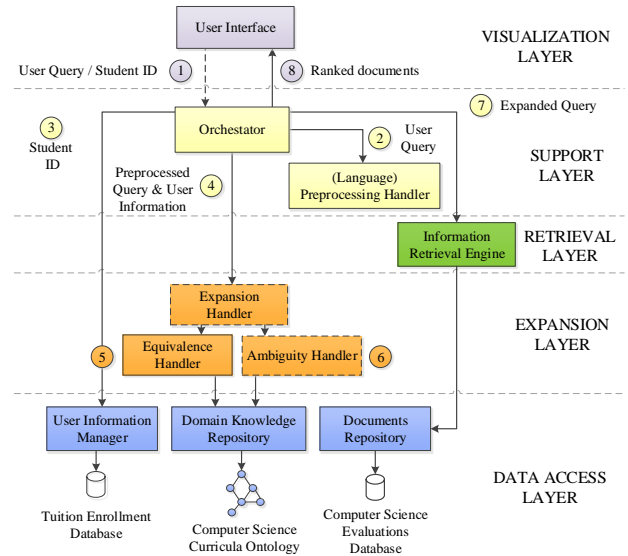


Fig. 1. Model architecture

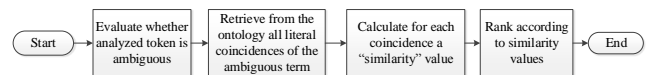


Fig. 2. General flow of the disambiguation mechanism

order to reduce different representations of a same concept to a single base form.

B. Expansion Handler

This component is in charge of performing the search, selection and addition of significant new terms to the query. It is supported by two sub-components which aim at dealing with two different NL features.

1) *Equivalence Handler*: Given that a same concept can be expressed by different words, this component aims to enrich the query by adding different but semantically equivalent words. Has as input the resulting terms from the preprocessed query. In first place, a list of nodes which does not contain the analyzed term will be retrieved. Then, a search of that term within the synonyms of the retrieved nodes will be performed, and if the term is found in the synonyms, both, synonyms and main concept, will be added to the expanded query.

2) *Ambiguity Handler*: This component will allow to identify the related concept of one or more words considered ambiguous within a knowledge domain. A domain ontology and the other supplementary query terms will support the identification of possible concepts to which the ambiguous term in analysis could be making reference. Figure 2 shows the general flow for the proposed mechanism, which was designed adapted from the general flow of the personal name disambiguation model [16]. The similarity computing consists of a recursive evaluation of the ontology nodes, and at each level evaluating if the supplementary query term is found in

the analyzed node or in its equivalent (*i.e.*, synonym nodes). For cases when the retrieved coincidence itself contains the supplementary term, the similarity is maximum (defined as 0), otherwise, the evaluation will continue in the hierarchically superior nodes which have a relationship with the analyzed (ambiguous term-coincident) node. The similarity value is defined as $CONSTANTEREC * LEVEL$, where level is the recursiveness level where the supplementary term was found. The more distant levels will be considered as less similar, and this will represent the end of the similarity calculation. Another stop condition is set to when reaching a maximum number of recursions without having found a coincidence of the supplementary term at any analyzed level. Finally, a ranking will be established based on the calculated scores. The top ranked term (similarity nearest to 0) represent the most accurate concept which will be added to the expanded query. The purpose of the $CONSTANTEREC$ constant will be described in the User Information Manager section.

C. Domain Knowledge repository

In order to represent and store the knowledge, a customized ontology in a university curricula in the CS domain was developed with the following considerations:

- A property `NombrePreferente` (Preferred Name), designed to store the principal name of each concept. Each node has only one preferred name.
- A property `Sinonimos` (Synonyms), which relates each node to their equivalent lemmatized terms without stop-words. Each node may have one or more synonyms.
- A property `Lemma`, which relates each node to their respective base form denomination. Each node in the ontology has exactly one associated lemma.

Even though each node's `NombrePreferente` is fulfilled based on the specialized knowledge provided by an expert, the lemmatized forms to be put in the other two properties are generated using the preprocessing mechanisms previously explained. After obtaining those lemmatized terms, they are manually inserted in the ontology development phase because the preprocessing mechanisms' overall execution time, when applied in a complex structure like this ontology is considerable, so online execution is not feasible.

D. Documents Repository

For simplification purposes, for this work it has been excluded from the scope the use of online information, and instead the documents repository consist of documents tagged with semantic content inserted to a relational DB.

E. User Information Manager

It consists of artifacts related to the information about the user. For our case the user information DB is a relational DB representing a tuition enrollment management system for university students, which contains information about the courses each student is currently enrolled in. It is within the disambiguation mechanism that user information will increase in relevance, because in case two or more concepts obtain

the same similarity score, the user information will be used to route the decision to one concept. In order to do so an additional flow is added to the similarity calculation which consist of verifying if the analyzed node is present in the user information and if so, decrementing the score in order to get it closer to a better similarity score. When using the user information the $CONSTANTEREC$ constant increase in relevance, because if the space enabled by the use of this constant would not exist, it may be the case that when decreasing the similarity score of a concept a new tie occurs.

IV. PROTOTYPE IMPLEMENTATION

In order to demonstrate the feasibility of the proposed model, we developed a prototype. The SRM was built based on the `StopAnalyzer` from Lucene library, and was configured with the default stopwords dictionary used by Lucene's `SpanishAnalyzer`. The LM was built based on the default Spanish lemmas dictionary from the Freeling language analysis tool. The disambiguation mechanism was built with support of the SPARQL query language for navigation and retrieval from the ontology. As part of the configuration for the prototype, the stop for the recursion was set to a deep of up to 5 levels, and the $CONSTANTEREC$ constant value was set to 3. The user information DB was filled with 30 tagged documents including university examinations in the CS domain and the IR Engine was developed based on Lucene without particular customizations. The developed ontology was created using Protégé in OWL/RDF.

V. RESULTS

As previously explained, the ontology was developed considering knowledge on a CS curricula. We took a query stating *pilas y quicksort*, in an attempt to retrieve information regarding to the specific topics of `Stacks` (abstract data type) and the `Quicksort` algorithm. The word *Pilas* is an ambiguous plural-form Spanish word. The word *Quicksort* even though it does have a Spanish translation, the reference to the sorting algorithm can be found indistinctly either in English or in Spanish. When the user inputs the query (fig. 1 step 1), the SRM removes the stopword *y* (and) and the LM converts the resulting *pilas quicksort* query to *pila* (single form) *quicksort* (step 2). Next, the student ID provided by the user is sent to the User Information Manager in order to retrieve the courses he is currently enrolled in (step 3). Then, the `Orchestrator` sends the preprocessed query to the `Equivalence Handler` (step 5) in order to verify if any of the preprocessed query terms is present in a synonym node. In this case, the word *Quicksort* was found as synonym of the main node *ordenamiento rapido*, for that reason this word was added as an expansion term. The relevance of this term ends here because the word `Quicksort` is not ambiguous. On the other hand, the word *pila* also goes through the mentioned mechanisms, without major relevance in respect to the `Equivalence Handler`. Continuing with the flow, the `Ambiguity Handler` (step 6) calculated the similarity values between the ambiguous token and the supplementary one. In this case, the `AplicacionesPilas` and

TADPilas nodes reached the same optimum because both of them include the term *pila*; however, inside the ontology both refer to different topics related to stacks. In this point the user information gains relevance for the disambiguation because in the ontology structure the *AplicacionesPilas* node belongs, by transitivity, to the *Algoritmia (AL)* course, which includes training in problems and applications of stacks, and the *TADPilas* node belong to the *Fundamentos de Programacion (Programming Fundamentals - PF)* course, which includes theoretical instruction on the Stack abstract data type. When previously consulted to the DB, it was determined the user is currently taking the AL course, but not PF. For that reason, the node from the AL course was selected as a better match. The final expanded query is *pila ordenamiento rapido aplicacion quicksort* which is sent to the IR Engine (step 7) in charge of retrieving the information and outputting the ranked list of documents (step 8).

When using the IR engine without QE, 6 documents were retrieved, all of them containing the word *pila* in their tags, and the ones in the top were mostly related to the theory of the abstract data type of the PF course. From the 6 documents, just two of them were relevant, which led to a 33% of precision. When using QE 10 documents were retrieved. The one in the top was indeed the most relevant which included in the same university examination exercises about applications of stacks and the quicksort method, and gradually other documents relevant to a lesser degree. From the 10 documents, 7 of them had a degree of relevance, which led to a 70% of precision.

VI. DISCUSSION

After the execution of tests we realized some stopwords would better be excluded from the stopwords list. This is the case of the word *no* from the query *no programacion en pascal* (not pascal programming). Even though the word *no* can be considered a stopword, its existence in order to keep the semantic integrity of the complete query is considered a relevant factor to take into account. However, for this work, those particular scenarios will not affect the results because the scope of this model excludes the analysis by propositional logic. We use the user information only for disambiguation cases, and don't directly include that information as terms for the expansion. That is because the information obtained regarding the user may be more general, and adding it as terms for the expansion in the total of cases could generalize the query and negatively affect the precision measure.

VII. CONCLUSION AND FUTURE WORKS

The developed tool based on the proposed model has proven that from an ambiguous query was possible to retrieve relevant information for the user. Different tests were performed in order to measure the tool. Tests without using QE resulted in an overall of 16.5% of precision, whereas tests using QE resulted in an overall of 69% of precision, so it reaffirms the proposed model led to better results. From a point of view of benefits, the selected domain is particularly useful for academic purposes, because the tool can be used for students

from careers related to CS to retrieve relevant information for their studies or research projects. In second place, this model is generic, so if the ontology is changed to another which follows the described structure, it is possible to perform the searching. In third place, the result of this work is a conceptual model which can be implemented on different platforms and without dependencies on specific technologies.

It would be interesting to test the model with other different domain ontologies. Regarding the mechanisms, it would be very useful one that pulls information from online sources and another that captures in an automatic way the always dynamic user information. Finally, we propose the inclusion of further query preprocessing which takes into account propositional logic, and so stopwords dictionary can consider those cases for accurate results.

REFERENCES

- [1] P. H. Cleverley and S. Burnett, "Retrieving haystacks: a data driven information needs model for faceted search," *Journal of Information Science*, vol. 41, no. 1, pp. 97–113, 2015.
- [2] M. C. de Andrade and A. A. Baptista, "Researchers' information needs in the bibliographic database: A literature review," *Information Services and Use*, vol. 34, no. 3, pp. 241–248, 2014.
- [3] Y. Gupta, A. Saini, and A. K. Saxena, "A new fuzzy logic based ranking function for efficient information retrieval system," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1223–1234, 2015.
- [4] M. Mitra and B. B. Chaudhuri, "Information retrieval from documents: A survey," *Information Retrieval*, vol. 2, no. 2-3, pp. 141–163, May 2000.
- [5] F. Ren and D. B. Bracewell, "Advanced information retrieval," *Electronic Notes in Theoretical Computer Science*, vol. 225, no. 0, pp. 303–317, 2009.
- [6] M. Kobayashi and K. Takeda, "Information retrieval on the web," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 144–173, 2000.
- [7] M. Mitra, A. Singhal, and C. Buckley, "Improving automatic query expansion," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '98. New York, NY, USA: ACM, 1998, pp. 206–214.
- [8] G. J. Hahm, M. Y. Yi, J. H. Lee, and H. W. Suh, "A personalized query expansion approach for engineering document retrieval," *Advanced Engineering Informatics*, vol. 28, no. 4, pp. 344–359, 2014.
- [9] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *International Journal of Human-Computer Studies*, vol. 43, no. 56, pp. 907–928, Nov. 1995.
- [10] H. Wang, Y. Guo, X. Shi, and F. Yang, "Conceptual representing of documents and query expansion based on ontology," in *Web Information Systems and Mining*, ser. Lecture Notes in Computer Science, F. L. Wang, J. Lei, Z. Gong, and X. Luo, Eds. Springer Berlin Heidelberg, Jan. 2012, no. 7529, pp. 489–496.
- [11] H. Melgar S., D. Salas Guillen, and J. Gonzales Maceda, "Ontology based inferences engine for veterinary diagnosis," in *Semantic Technology*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 79–86.
- [12] S. Kara, . Alan, O. Sabuncu, S. Akpınar, N. K. Cicekli, and F. N. Alpaslan, "An ontology-based retrieval system using semantic indexing," *Information Systems*, vol. 37, no. 4, pp. 294–305, 2012.
- [13] M. Fernandez, I. Cantador, V. Lpez, D. Vallet, P. Castells, and E. Motta, "Semantically enhanced information retrieval: An ontology-based approach," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 9, no. 4, pp. 434–452, 2011.
- [14] V. Jalali and M. Borujerdi, "The effect of using domain specific ontologies in query expansion in medical field," in *International Conference on Innovations in Information Technology, 2008. IIT 2008*, Dec. 2008, pp. 277–281.
- [15] W. Ali and S. Khan, "Ontology driven query expansion in data integration," in *Fourth International Conference on Semantics, Knowledge and Grid, 2008. SKG '08*, Dec. 2008, pp. 57–63.
- [16] Z. Lu, Z. Yan, and L. He, "OnPerDis: Ontology-based personal name disambiguation on the web," in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, vol. 1, Nov. 2013, pp. 185–192.

Toward an Architecture for Model Composition Techniques

Kleinner Farias, Lucian José Gonçalves,
Murillo Scholl, Maurício Veronez

PIPCA, University of Vale do Rio dos Sinos (Unisinos)
São Leopoldo, RS, Brazil
kleinnerfarias@unisinos.br,
lucianjosegoncales@gmail.com,
murillosholl@hotmail.com, veronez@unisinos.br

Toacy Oliveira

PESC/COPPE, Federal University of Rio de Janeiro
(UFRJ)
Rio de Janeiro, RJ, Brazil
toacy@cos.ufrj.br

Abstract—Academia and industry are increasingly concerned with producing general-purpose model composition techniques to support many software engineering activities, e.g., evolving UML design models or reconciling conflicting models. However, the current techniques fail to provide flexible and reusable architectures, a comprehensive understanding of the critical composition activities, and guidelines about how developers can use and extend them. These limitations are one of the main reasons why state-of-the-art techniques are often unable to aid the development of new composition tools. To overcome these shortcomings, this paper, therefore, proposes a flexible, component-based architecture for aiding the development of composition techniques. Moreover, an intelligible composition workflow is proposed to help developers to improve the understanding of crucial composition activities and their relationships. Our preliminary evaluation indicated that the proposed architecture could support composition tools for UML class, sequence, and component diagrams.

Keywords: *model composition, architecture, UML*

I. INTRODUCTION

Researchers and practitioners recognize the importance of model composition in many software engineering activities [1][2][8], e.g., evolving design models to add new features and reconciling multi-view models developed in parallel by different software development teams [5][7]. In collaborative software development, for example, separate virtual teams may concurrently work on a partial model of the overall architecture to allow developers to concentrate more effectively on parts of the architecture relevant to them. At some point, it is necessary to bring these models together to generate a “big picture” view of the overall architecture. Unfortunately, this composition task is considered as an error-prone, time-consuming task [1][8]. In [8], the authors highlight that the model comparison and merging task are tedious, time-consuming, and error-prone. In [1], Mens reinforces that software merging continues to be “*a time-consuming, complicated, and error-prone process because many interconnected elements are involved and merging depends on both the syntax and semantics of these elements.*”

For this reason, there has been a significant body of research into defining model composition techniques in the areas of software modeling [9], synthesis of feature models

[12], and software product lines [11]. In fact, the high number of conventional, general-purpose composition techniques created in the last decade attests this importance, e.g., Kompose [13], IBM Rational Software Architect (IBM RSA) [4], MATA [3] and Epsilon [5].

Model composition can be briefly defined as an operation where a set of tasks should be performed over two input models, M_A and M_B , in order to produce an output-intended model, M_{AB} . While M_A represents the base model, M_B consists of the delta model having all increments that should be inserted into M_A to transform it into M_{AB} . Existing composition techniques usually produce an output composed model (M_{CM}) that often does not match the output intended model (M_{AB}), i.e. $M_{CM} \neq M_{AB}$. Because the elements of the input models usually conflict with each other in some way, and these techniques end up being unable to deal with all contradicting changes properly.

The problem is that the general-purpose feature of composition techniques hinders coping with a set of particular composition cases. Unfortunately, they fail to provide flexible, reusable architectures, a comprehensive understanding of the chief composition activities, or even provide guidelines about how developers can use and extend them. The limitations can be explained for two principal reasons as follows: (1) composition techniques are not structured with design-for-change principles upfront, being rigid to support modern composition strategies. Typically, developers are commonly forced to go through the source code to locate the component to-be changed or even create new architectural components to implement upcoming features. An incorrect modification of such components can jeopardize the implementation of new features, and (2) they rely on generic representation, i.e., usually graph, rather than on the semantics of constructs of OO design modeling languages, e.g., UML [9]. Since the current multi-view UML diagrams demand different but complementary ways to be integrated, generic approaches tend to produce output models with inconsistencies. Consequently, they fail to provide a systematic and flexible way to derive composition techniques for a particular purpose, or even provide guidelines about how developers can evolve them.

To overcome these shortcomings, this paper, therefore, proposes a flexible, component-based Architecture for aiding the development of Model Composition Tools, hereafter called

MoCoTo-Arch, and a model composition workflow for helping developers to improve the understanding of the crucial composition activities and their relationships. Our preliminary evaluation indicated that the proposed architecture could support the development of composition tools for UML class, sequence, and component diagrams.

The remainder of the paper is organized as follows. Section II contrasts this work with the current literature. Section III presents the MoCoTo architecture. Section IV describes the composition tool developed using the MoCoTo-Arch. Finally, Section V presents some concluding remarks and future work.

II. RELATED WORK

The last few years, some techniques have been proposed, including MATA [3], a tool based on graph transformations for composing aspects, IBM RSA [4], a robust software modeling and model composition tool, and Epsilon [5], an Eclipse Plugin consists of a family of languages for composing models and other vast functions. Although some works provide programming languages to express composition logic [5], little is known about the flexibility and capacity of the current techniques to support new composition strategies. This lack hinders the understanding about how such techniques can evolve to support the composition of new design models, until then not supported.

Many works aim at studying the proactive detection and earlier resolution of composition conflicts. In [2], Brun *et al.* proposes Crystal, an approach to help developers identify and resolve conflicts early. The authors highlight the ever-present occurrence of composition conflicts, more than would be expected, e.g. overlapping textual changes and their subsequent build and test failures. Likewise, Sarma [6] comes up with Palantir, a workspace-aware approach for detecting and resolving contradicting changes in early stage. Although these two approaches are interesting studies, they neither propose a flexible, design-for-change approach nor provide a comprehensible workflow to leverage the understanding of the inherent model composition activities and its relationships. Still, they overlook the challenging considering the synthesis of heterogeneous design models, thus not leading to broader generalizations of their findings at the modeling level.

On the other hand, in [7], the authors discuss the problem of tolerating conflicts and transforming them in an object of enhancement in collaborative software development. The purpose is to maintain all the conflicting changes in the resulting model. For this, they propose annotations, relating the conflicts as well as the developers involved in a further resolution. In [8], the authors introduce an approach to find similarities between business process models. For this, they define metrics to match the input model elements, and use typography and synonym dictionary.

III. MoCoTo ARCHITECTURE

We present the MoCoTo's built-in model composition process by identifying the phases, the artifacts generated, and the main activities required to transform the input models, M_A and M_B , into an intended output composed model, M_{AB} . Moreover, it details the most relevant characteristics related to design and implementation issues, including feature model

elicited, components that implement such features, and the architectural design.

A. Model Composition Process

Figure 2 shows the proposed model composition process. It is represented as an intelligible workflow, thus allowing developers to understand the inherent activities of a composition process in terms of phases, its artifacts, activities and the flow among each other.

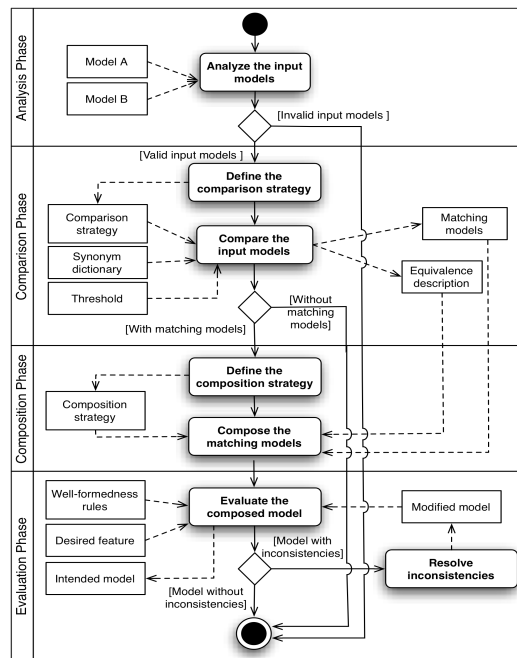


Figure 1. The proposed model composition process.

1) *Analysis Phase*: the prime goal is to analyze the input models adequately as a basis for assuring the composition of compatible input models as well as preventing input models with inconsistencies. This phase should attend to the *Lifecycle Analysis Milestone* criteria answering: are the input models of the same type? Do the input models have inconsistencies? If the input models do not attend to this milestone, the composition process can be cancelled or repeated after the input models are redesigned to comply with milestone criteria.

2) *Comparison Phase*: the chief goal is to systematically compare the input models for determining the similarity between their elements, thereby mitigating mistaken equivalence relationships, including false-positive and false-negative ones. The MoCoTo architecture supports a range of matching strategies (but not limited to), including default, partial and complete one [10], to alleviate the more severe risk items. The inputs of this phase are: ngram algorithm, synonym dictionary, matching strategies, matching rules, and threshold. Hence, producing the following outputs: (1) the similarity matrix, specifying the degree of equivalence (ranging from 0 to 1) between the input model elements; (2) the matching elements, a description of the elements of M_A and M_B being considered equivalent; (3) the no-matching elements, a description of the elements of M_A and M_B being considered no equivalent. Two input elements are considered similar when

the degree of similarity between them is equal or higher than 0.8, the threshold used. This threshold is based on previous studies [10] on model comparison, which have demonstrated its usefulness.

3) *Composition Phase*: The master goal is to carefully bring together the matching and no-matching elements for producing an output intended model, M_{AB} . For this, the proposed composition technique takes into account the similarity matrix, as well as the description of the matching and no-matching elements of the input models. In addition, it uses a range of well-established composition strategies (but not limited to), including *override*, *merge*, and *union* [15], to accommodate the elements from M_B into M_A , thereby alleviating the more severe risk items. The MoCoTo's built-in composition strategies integrate the matching elements while the no matching ones are just inserted into the M_{AB} . Thus, M_{AB} represents the matching and no-matching elements, all blended systematically.

4) *Evaluation Phase*: the master goal is to evaluate if the output model produced in the previous phase matches the output intended one, i.e. $M_{CM} = M_{AB}$. If $M_{CM} \neq M_{AB}$, then M_{CM} needs to be manipulated so that the inconsistencies can be resolved. For this, the tool checks if the output model is in compliance with well-formedness rules defined in the UML metamodel and meets a set of desired features specified by the user. If the model has inconsistencies, then some transformation rules can be applied to transform M_{CM} into M_{AB} . This phase end producing the output intended model. After detailing the composition process, the next Section focuses on describing the design and implementation issues required to put the process in practice.

B. MoCoTo architecture feature model

The MoCoTo architecture was proposed due to several reasons and requirements identified in previous works [10][14][15]. First, our experience with model composition has indicated the increasing need for reusable architecture to support and guide the development of new composition tools. Second, it is representative of the model composition domain, since its design decomposes the key concerns into well-modularized features. Third, it assures the derivation of different products by defining several variability points related to heterogeneous strategies related to analyzing, comparing, and composing the input models. Lastly, it allows evaluating the models generated and persisting the results. Thus, the proposed architecture provides a set of pivotal features, including analysis of the input models, comparison of the input

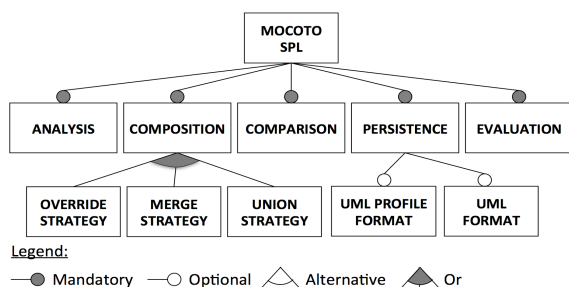


Figure 3. A simplified MoCoTo-Arch feature model.

models, composition of the equivalent input model elements, persistence of the output model generated, and evaluation of the output model.

Figure 3 shows a simplified view of MoCoTo-Arch's feature model. Thus, to develop composition tools developers should firstly implement the *mandatory features*, including analysis, comparison, composition, persistence, and evaluation. Besides identifying a set of core functionalities, the mandatory features seamlessly specify their dependencies in an easy-to-understand manner. An ever-present concern throughout the MoCoTo architecture was to assure the mandatory features comply with the model composition process described in Figure 3, for example, the *analysis* feature implements the first phase and the *persistence* feature provides the functionality required to persist the output-composed model generated at the end of the model composition process. The *optional features* are the types of file format that the output-composed model can be persisted, including UML and UML profile format. The *or features* are represented by the composition strategies, and the comparison strategies, the latter are not shown in the feature model for space constraints. Thus, one (or more) comparison and composition strategy should be selected when a composition tool is derived from the MoCoTo-Arch.

C. MoCoTo architectural components

Figure 4 shows the components that are responsible for implementing the feature model as well as relates them with the features depicted in Figure 3. The small squares located on the left or bottom sides of the components represent this feature-component mapping. For instance, the C on the top of the Comparison component (Figure 4) indicates that this component contributes to the implementation of the comparison feature. This *design-for-features* is supported by the component-based development, a systematic feature-component mapping and aspect-oriented programming.

This method of decomposing components based on the features allows creating autonomous, well-modularized design elements within a model composition tool, thereby promoting the reuse of previously elicited feature and constructed components. Each component was designed to: (1) be a self-contained module that encapsulates the state and behavior of a set of executable elements, which are responsible for the implementation of one (or more) feature; (2) present emergent behaviors resulting from the interaction of its executable elements, i.e., one or more classes that realize the expected functionalities of the features; and (3) have well-defined

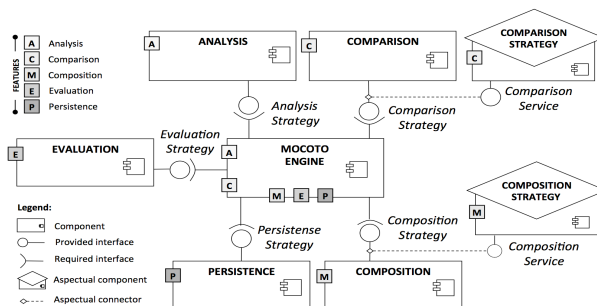


Figure 4. The MoCoTo architectural components.

interfaces, including the provided and required ones. For example, to provide the behavior of matching two input models, the `Comparison` component implements the provided interface, `ComparisonStrategy`. If new components are inserted, then they should implement this interface only. Moreover, Figure 4 focuses on presenting the components as a coherent group of elements implementing one (or more) feature. Each component can be seen as a building block that plays a crucial role within the model composition process.

D. MoCoTo multilayered architecture layers

The logical, multilayered architecture enables us to support a well-modularized design, thereby putting the heterogeneous, crosscutting concerns, previously described in Figure 2, in shape. The architecture is composed by five layers: (1) *Presentation layer* represents the topmost tier of the application gathering the input data required to perform the functionalities and putting out the results to the compositions; *Application layer* encompasses MoCoTo's engine and its operators. It is responsible for orchestrating, along with its operators, the composition process as a whole. As an orchestrator, it plays a pivotal role by providing the principal main entry point, coordinating incoming composition requests, transforming the requests into commands for the operators, and rendering views; (3) *Variability layer* implements the variation points. For this, aspectual components weave the behaviors (or advices) from design elements (from the business logic layer) to the operators (in the application layer). Aspectual components augment the operators with additional or alternative behaviors, i.e. strategies and their rules; (4) *Business Logic layer* defines a family of algorithms that implement the MoCoTo features. These algorithms analyze the input models, seek to find the commonalities and differences between the input models, integrate the commonalities, and then evaluate the output models, and (5) *Infrastructure layer* accommodates the concerns related to handling exception, data access, persistence and logging, which are key crosscutting functionalities to put the composition process in practice.

IV. CASE STUDY

We evaluate this work by implementing a model composition tool based on the MoCoTo architecture. The tool, so-called MoCoTo, is an Eclipse Plug-in that allows a seamless integration with Eclipse Platform. In addition, it makes use of a range of Eclipse modeling technologies, including EMF, UML2, GEF, UML2 tool, to implement all required activities described in the model composition process discussed earlier. MoCoTo ties together these technologies in such a way that makes it easy to use, even for users with little or no Java or XML coding experience. For example, UML2 API reads and filters information from the tags of files written in XML and transforms it to an abstract data model in which input model elements can be manipulated as objects.

V. CONCLUSIONS AND FUTURE WORK

This paper introduced a flexible, component-based architecture for supporting the development of model composition techniques, and an intelligible model composition workflow for aiding developers to comprehend the crucial

composition activities and their relationships more properly. We also reported the MoCoTo tools, a composition tools defined over the MoCoTo-Arch. The preliminary results have indicated that the proposed architecture is able to support the development of composition tools for UML models. Although MoCoTo-Arch has shown to-be useful, further empirical studies are still required, other than case study presented, to check their usefulness to compose other models, including business process models, and with different developers, compared to other composition techniques.

The future investigations should seek to answer some questions such as: (1) do developers invest significantly more effort to develop a new composition technique than derive one from MoCoTo-Arch? (2) How effective is MoCoTo to combine realistic, semantically richer design models? (3) Do developers invest more effort to resolve semantic inconsistencies than syntactic ones using a strategy-based composition technique? (4) How do developers observe the benefits of the composition process? Lastly, this work represents a first step in a more ambitious agenda on better supporting the elaboration of model composition techniques.

ACKNOWLEDGMENT

This work was funded by Universal project – CNPq (grant number 480468/2013-3).

REFERENCES

- [1] T. Mens, "A state-of-the-art survey on software merging," *IEEE Trans. Softw. Eng.* 28(5), 449–562, 2002.
- [2] Y. Brun et al., "Proactive Detection of Collaboration Conflicts," In: 8th SIGSOFT ESEC/FSE, pp. 168-178, Szeged, Hungary, 2011.
- [3] J. Whittle, P. Jayaraman, "Synthesizing hierarchical state machines from expressive scenario descriptions," *ACM TOSEM*, 19(3), 1–45, 2010.
- [4] IBM Rational Software Architecture (IBM RSA), <http://www.ibm.com/developerworks/rational/products/rsa/>, 2011.
- [5] Kolovos et. al., "The Epsilon Book," <http://eclipse.org/epsilon/doc/book/>, 2015.
- [6] A. Sarma et al., "Palantir: early detection of development conflicts arising from parallel code changes," *IEEE TSE*, vol. 99, no.6, 2011.
- [7] K. Wieland et al., "Turning conflicts into collaboration - concurrent modeling in the early phases of software development," *CSCW: The Journal of Collaborative Computing*, 22 (2013), 2-3; 181 - 240.
- [8] M. La Rosa et al., "Business process model merging: an approach to business process consolidation," *ACM TOSEM*, 22(2): 11, 2013.
- [9] OMG, UML: Infrastructure version 2.4, August 2011.
- [10] K. Farias et al., "A flexible strategy-based model comparison approach: bridging the syntactic and semantic gap," *Journal of Universal Computer Science*, 15(11):2225-2253, 2009.
- [11] P. Jayaraman, J. Whittle, A. Elkhodary, H. Goomaa, "Model Composition in Product Lines and Feature Interaction Detection using Critical Pair Analysis," *MODEL'7*, pages 151-165, 2007.
- [12] S. She, U. Ryssel, N. Andersen, A. Wasowski, K. Czarnecki, Efficient synthesis of feature models, *Information & Software Technology*, 56(9): 1122-1143, 2014.
- [13] Fleurey et. al., Kompose : A generic model composition tool, <http://www.kermeta.org/kompose/>, 2015.
- [14] S Clarke, Composition of Object-Oriented Software Design Models, Ph.D. Thesis, Dublin City University, January, 2001.
- [15] K. Farias, Empirical Evaluation of Effort on Composing Design Models, PhD thesis, Department of Informatics, PUC-Rio, Rio de Janeiro, RJ, Brazil.

JSAN: A Framework to Implement Normative Agents

Marx Viana¹, Paulo Alencar³, Everton Guimarães², Francisco Cunha¹, Donald Cowan³, Carlos Lucena¹

¹Pontifical Catholic University - PUC-Rio – Rio de Janeiro, RJ - Brazil
mleles@inf.puc-rio.br, fplacido@inf.puc-rio.br, lucena@inf.puc-rio.br

²University of Fortaleza – Unifor – Fortaleza, CE - Brazil
eguimaraes@unifor.br

³University of Waterloo – Waterloo, Ontario - Canada
palencar@uwaterloo.ca, dcowan@uwaterloo.ca

Abstract— Norms have become a promising mechanism to ensure that open multi-agent systems (MASs) produce a desirable social outcome. MASs can be defined as societies in which autonomous agents work to achieve both societal and individual goals. Norms regulate the behavior of agents by defining permissions, obligations and prohibitions, as well as encouraging and discouraging the fulfillment of norms through rewards and punishments mechanisms. Once the priority of software agent is the satisfaction of its own desires and goals, each agent must evaluate the effects associated to the fulfillment or violation of one or more norms before choosing which one should be complied. This paper introduces a framework for normative MASs simulation that provides mechanisms for understanding the impact of norms on an agent and the society to which an agent belongs.

Keywords. Normative Agents; Multi-agent Systems, Simulation; Norms.

I. INTRODUCTION

Open multi-agent systems (MASs) are societies in which autonomous, heterogeneous and independently designed entities work towards specific goals [9]. In order to deal with autonomy and diversity of interests among the different members, such systems provide a set of norms that is used as a social control mechanism to ensure that a desirable social order in which agents can work is maintained [19]. For the best of our knowledge, norms can be defined as agent-oriented mechanisms for regulating the behavior of agents through the definition of obligations (agents must accomplish a specific outcome), permissions (agents can act in a particular way) and prohibitions (agents must not act in a specific way) [13]. Although norms are promising mechanisms to regulate the behavior of agents, one should take into account that they are autonomous and, therefore, free to decide to fulfill or violate each system norm. This type of agent reasoning refers to normative strategies [10].

Several approaches [4, 19] have been proposed for the specification and implementation of norms. According to Garcia Camino *et al.* [4], norms constitute a powerful coordination mechanism among heterogeneous agents. The authors propose means to specify and explicitly manage the normative constraints on agents (i.e., permissions, prohibitions and obligations), with which distinct deontic notions and their relationships can be captured. Silva presents a normative language to specify norms and proposes

the implementation of such norms by using a rule-based system [19]. The implementation is achieved by automatically transforming the specification of each norm of the system into a set of rules used to govern the behavior of the agents according to the norm. In addition, these works have focused on the definition of parts of an infrastructure that can be used by Belief-Desire-Intention (BDI) agents, which consists of beliefs, desires and intentions as mental attitudes that deliberate human action [14] to reason about norms [7, 11]. However, there is still a need to define an agent-oriented framework to support the implementation of goal-oriented normative agents. The main purpose of goal-oriented normative agents consists on achieving their goals and desires while satisfying system norms. Although there are a number of existing agent-oriented platforms such as [1, 5, 11, 14], there is a lack of support of a framework for normative agents.

In this context, we present a framework for Normative Agent Java Simulation (JSAN). This framework aims at providing support to build and operate agents able for dealing with goals, desires and norms – that is, agents that support normative reasoning. JSAN extends the JASON framework [1], which already provides support for the implementation of BDI agents and a set of hot-spots that enable the implementation of normative functions. By using these new functions, it is possible to build BDI agents that can check whether they should: (i) adopt a norm, (ii) evaluate the effects on their desires with respect to the fulfillment or violation of a norm, (iii) detect and solve conflicts among norms, and (iv) select desires and plans based on the decision on whether to fulfill a norm. A preliminary overview of the framework is described in [20]. The remainder of this paper is organized as follows. Section 2 focuses on the representation of norms, while Section 3 presents the JASON Platform. Section 4 discusses related work. In Section 5 the JSAN framework is detailed and Section 6 describes a case study by showing how agents deal with norms in real situations. Finally, Section 7 presents our conclusions and future work.

II. REPRESENTATION OF NORMS

According to Lopez [9], norms are designed to regulate the behavior of agents, and therefore, a norm definition should include the address of the agent being regulated. When the norm need be applied, the nature of the norm (permission, obligation or prohibition), as well as the

consequences of either fulfilling or violating the norm (reward or punishment) should be described. In this work, we use the norm representation described in [18], which is composed the representation of an element *norm* – it contains many different properties. Those properties are briefly described in Table 1. For example, the property *Addressee* is used to specify the agents or roles responsible for fulfilling the norm.

In order to understand the definition of norms and their representation better, imagine a Fireman Commander agent is leading the rescue of civilians who are in hazardous areas. This agent is responsible for regulating the behavior of all fireman agents and the usage of the resources available to them (e.g., helicopters, troops and land-based helicopters) – we are assuming the resources are limited. In addition, each fireman agent should perform a rescue according to specific norms. Eventually, a norm is sent to each fireman agent with the following state: “protect lives of civilians in hazardous areas.” This norm has the following attributes: (i) the addressees are the firemen agents; (ii) the required deontic concept is obligation; (iii) when an agent agrees to a norm that agent will receive a reward. In this case, the reward could be either air or ground transportation during the agent’s mission. If the fireman agent does not follow a certain norm directed at him in the environment, after violating the norm, this agent receives the punishments associated with the norm. For example, there are situations when a fireman agent requests aircraft support to accomplish a specific rescue operation activity that places him or her in a degree of risk above the one allowed by the norm. In this case, a punishment (e.g. a warning or an order that he should be temporarily restricted to headquarters to assist other rescuers) associated with the norm will be applied for the agent. Note the norm is activated if there is any person in a risky situation. In turn, the norm expires when all civilians are safe, and the state or element regulated by the norm is the action of using aircraft, because of the costs.

TABLE I. NORM ELEMENTS

Property	Description
Addressee	It is the agent or role responsible for fulfilling the norm.
Activation	It is the activation condition for the norm to become active.
Expiration	It is the expiration condition for the norm to become inactive
Rewards	It represents the set of rewards to be given to the agent for fulfilling a norm.
Punishments	It is the set of are the punishments to be given to the agent for violating a norm
DeonticConcept	It indicates if the norm states an obligation, a permission or a prohibition.
State	It describes the set of states being regulated.

III. THE JASON PLATFORM

The JASON platform enables the development and implementation of Belief, Desire and Intention (BDI) agents. In addition, the platform uses a language called AgentSpeak for implementing agents. Figure 1 illustrates how JASON interprets AgentSpeak programs [12]. In this figure, sets of beliefs, events, plans and intentions are represented by

rectangles. Diamonds represent the selection of an element of a set and circles represent some of the processes involved in the interpretation process.

Each interpretation cycle updates the list of events based on the agent’s perception of the environment, the messages the agent receives and the information coming from the agent’s own execution of a plan. The *Belief Review Function* (BRF) revises the *Belief Base* with a literal to be added or deleted, and the intention structure required to change the belief. A single event is chosen by the *Event selection function* (SE) and the event is unified with the triggering events in the heads of plans by the *Unify Event* cycle that generates a set of all relevant plans. The context of such plans is verified according to the *Belief Base* by the *Check Context* cycle, which generates a set of options. The *Option Select Function* (SO) selects a single applicable option from the set of options, which becomes the intended means for handling the selected event. The option either pushes the plan on top of an existing intention (if the event was an internal one), or creates a new intention in the set of intentions (if the event was external, i.e., generated from perceptions of the environment). The *Intention Select Function* (SI) selects one of the agent’s intentions and this intention is executed by the *Execute Intention* cycle. When all formulas in the body of a plan have been executed, the whole plan is removed from the intention list, and so is the achievement goal that generated the plan. This ends a cycle of execution, and the interpretation starts over again, checking the state of the environment after agents have acted upon it and generated the relevant events.

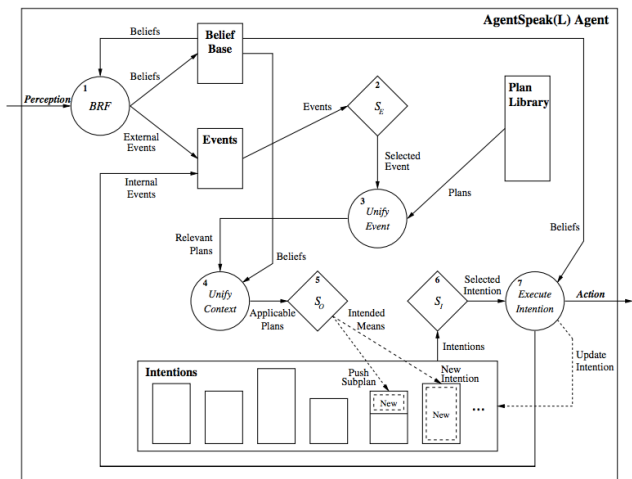


Figure 1. An Interpretation Cycle of an AgentSpeak Program [12].

IV. RELATED WORK

Some approaches [3, 8, and 10] have been proposed in the literature to develop agents that evaluate the effects of fulfilling or violating norms. For instance, the n-BDI architecture defined by Criado *et al.* [3] presents a model for designing agents capable of operating in environments governed by norms. Basically, the architecture selects objectives to be performed based on the priority associated with each objective. An objective’s priority is determined by the priority of the norms governing a specific objective. However, it is not clear in this approach how the properties

of a norm can be evaluated. In addition, the approach does not support a strategy to deal with conflicts between norms.

In turn, Meneguzzi and Luck [10] proposed a formal model using the Z specification language, for modeling agents that achieve their objectives based on the systems' norms. For instance, an agent created from such a model should be able to: (i) check if it is the one responsible for fulfilling a norm; (ii) verify the activation and expiration of a norm based on the beliefs of the agent; (iii) evaluate and decide to fulfill or violate every norm of the system; and (iv) make the decision to fulfill or violate a norm while removing or adding agent goals. Besides not showing how the evaluation of a norm is performed, the authors do not focus on (i) identifying and resolving of conflicts between norms; (ii) checking fulfilled or violated norms; and (iii) showing the influence of norms on the plan selection process and intentions of the agents.

Finally, Lopes *et al.* [8] defined a set of strategies that can be adopted by agents to deal with norms as follows: *Pressured*, *Opportunistic* and *Selfish*. For instance, the *Pressured* strategy happens when agents fulfill the norms to achieve their individual goals considering only the punishments that will harm them. Another strategy is the *Opportunistic* strategy, in which agents consider only the effects of rewards on their individual goals, and seek to fulfill only the norms for which the rewards of the individual goals are more important than those of the social goals. Finally, the *Selfish* strategy is the combination of the *Pressured* and *Opportunistic* strategies. Although this work provides some mechanisms for agents that handle standards, the authors provide a framework that can be extended to create simulations of normative multi-agent systems by including new strategies. In addition, this work cannot extend mechanisms to collect information during the simulations or mechanisms for generating norms and goals of the agents.

V. JSAN – A FRAMEWORK FOR NORMATIVE AGENT JAVA SIMULATION

This section describes the main concepts required to understand the framework proposed in this paper. In addition, we provide an overview of JSAN framework and discuss the different its different components, including the kernel (frozen-spots) and flexible points (hot-spots) [21].

A. The Framework

The JSAN framework is implemented using software agents, as illustrated in Figure 2. The JSAN extends the JASON framework [1], which is an interpreter for an extended version of the AgentSpeak language [15]. For the best of our knowledge, the AgentSpeak language is an agent-oriented programming language that supports the creation of BDI agents. In addition, JSAN framework provides a platform for development of multi-agent systems (MASs). The framework comprises three main functions: (i) agents responsible for dealing with norms; (ii) visualizations for agent simulations involving norms; and (iii) creation of norms. In order to implement normative agents, it is necessary to instantiate the JSAN framework. For extending

the framework, developers should implement the agents' goals, movement strategies (different agents' movement strategies) and normative strategies (to represent how the agents deal with norms). JSAN already supports a normative process composed of four activities, as detailed in Section IV.B.

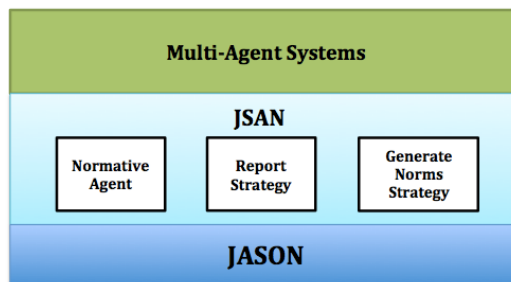


Figure 2. The JSAN Architecture.

B. The JSAN Architecture

The JSAN framework supports creating simulations that show the impact of norms on normative agents. Thus, the JSAN framework provides means for the implementation of Normative Agents (see Figure 3). For this purpose, there is a need to: (i) create the goals of an agent; (ii) create different movement strategies of an agent; and (iii) provide normative strategies to represent how the agents deal with norms.

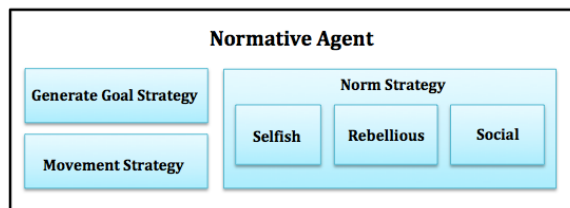


Figure 3. The Internal Structure of the Normative Agents provided by JSAN.

JSAN provides a normative application process to agents capable of reasoning about norms, represented by the *NormStrategy* class, which is composed of four activities (see Figure 4): *Norm Awareness* (Section IV.B.1), *Norm Adoption* (Section IV.B.2), *Norm Deliberation* (Section IV.B.3) and *Norm Impact* (Section IV.B.4).

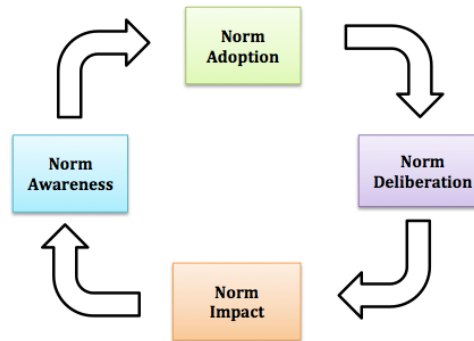


Figure 4. The Normative Strategy provided by the JSAN Framework.

The normative application process is based on the process proposed in [10]. Note that, although JSAN already

provides such process, it is possible to define alternative processes using the *NormStrategy* class. In addition, it is possible to implement different activities (or steps for the process) by extending the class *NormStrategy*. Once an active norm is defined in the environment for a specific software agent, some steps should be performed in the process in order to apply the norm. Each of those steps is described in the following.

1) *Norm Awareness* - the software agent identifies which norms are active in the environment, and hence, those norms are assigned to specific agentes.

2) *Norm Adoption* - the agents recognise their responsibilities towards other agents by internalising the norms where their responsibilities are specified.

3) *Norm Deliberation* - in order to execute a specific norm, an agent should access to diferent information: (i) the goals that should be hindered by satisfying the normative goals; and (ii) the goals they might benefit from the associated rewards.

4) *Norm Impact* - after the agents execute the norm (step 3), the goals of the agents will be updated. After that, the cycle continues and the agent starts identifying other norms that should be addressed.

It is important to note the steps need to take into account not only the agents' goals, but also the mechanisms available to avoid the violation of norms (e.g. rewards and punishments). That is, agents should consider the so called social pressure (i.e., agents recognized their responsibilities before other agents), of norms before making any decision regarding norms.

C. Hot-Spots and Frozen-Spots

As previously mentioned, JSAN is an extension of JASON framework, and therefore, they share the same core and hot-spots. The process used for the communication between software agents, and the identifiers of agents are examples of hot-spots that JSAN inherited from JASON. In addition, JSAN define other specific hot-spots, which are:

Environment (*EnvironmentSimulation* class): The *ExecuteAction* method was extended from the Environment. Whenever an agent tries to perform an essential action, the agent's identification and its chosen actions are passed to this method. For this reason, the *ExecuteAction* method must verify that the action is valid and then perform the action. The action will possibly change an agent's perception. If this method returns *true* it means that the action was performed successfully.

Normative Agent (*NormativeAgent* class): By extending such a class and implementing the *execute* method it is possible to define different algorithms to execute the plans of an agent.

Created Norms (*GenerateNormsStrategy* class): It is possible to define new strategies to create norms in the environment.

Norm Strategies (*NormStrategy* class): It is possible to define new strategies (or plans) for agents to deal with

norms, and proceed to execute the activities of the normative application process. JSAN already provides a default process implemented in the classes *Selfish*, *Rebellious* and *Social*.

Created Agent Goals (*GenerateGoalStrategy* class): It is possible to create new goals for agents.

Movement Strategies (*MovementStrategy* class): It is possible to create new movement strategies for the agents in the environment.

Simulation Environment Report (*ReportStrategy* class): It is possible to define reports as the output of the simulation. In order to use this mechanism it is necessary to extend the *ReportStrategy* class, where the following parameters should be provided: (i) the environment of the simulation is being carried out; and (ii) an object of *NormStrategy* class, which contains the strategy used by the agent to handle the norms

VI. USAGE SCENARIO: EVACUATING PEOPLE FROM AREAS OF RISK

The need to build platforms to assist both experts on risk analysis as well as experts in planning the evacuation of civilians located in areas of risk is currently a critical problem, especially in large cities that have grown in an unplanned and often disorderly manner [2]. These cities experience many circumstances in which people need to be rescued from dangerous areas as a result of floods, landslides and other natural phenomena. Landslides, for example, are difficult to predict since they depend on many factors such as climate, soil properties, and humidity and the specific relationship between them. The annual number of landslides is estimated to be in the thousands, and the associated infrastructure damage resulting from them is worth billions of dollars [16]. Planning evacuations from areas of risk can be assisted by simulations using the JSAN framework. These simulations implement scenarios representing hazardous situations. For example, a simulation can be used as a way to examine different strategies that can be adopted by the firemen agents, who are regulated by norms, in order to rescue civilian personnel.

A. Overview

The implemented simulation is composed of firemen agents, civilian agents and norms, as illustrated in Figure 5. The goal of the firemen agents is to rescue people who are at risk. The civilian agents do not deal with norms. The structure of the norms is created to extend the JSAN *GenerateNormStrategy* class. This structure was proposed in [17]. These simulations are normative multi-agent systems that receive data containing information about (i) a hazardous area, such as weather conditions, (ii) civilian personnel, (iii) ways of saving civilians by removing them from these locations (with troops, land vehicles or aircraft), (iv) norms that firemen agents must follow during the rescue operation, and (v) rescue plans used in the simulation. Through the simulations, it is possible to find different solutions the firemen agents can follow in order to evacuate civilians from these hazardous areas.

To deal with these problems and understand the related norms, the fire-fighter agents have: (i) a set of objectives that

is connected directly to their individual satisfaction; (ii) an information base of facts collected by the simulation environment to help characterize risky locations; and (iii) a base of strategies used to deal with norms. The Selfish strategy of the firemen agents (See description in Section III) was implemented using the calculate method of the Selfish class (see Figure 3). This method aims at analyzing the situation where norm compliance will help the agent to meet its individual goals, without forgetting the social goals, since norm compliance is directly linked to the benefits the agent will receive.

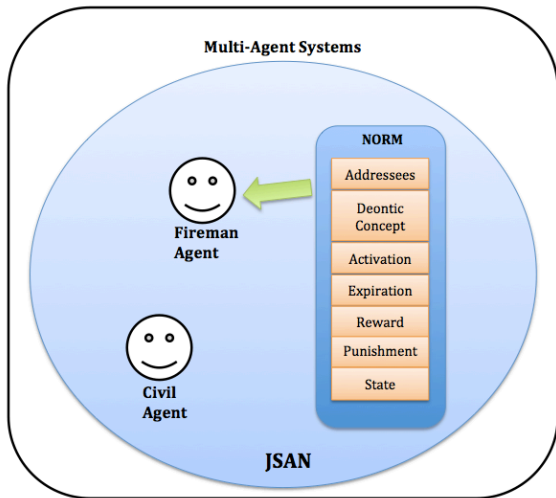


Figure 5. Conceptual model of the usage scenario.

We used the *PlanGenerateNorms* class, an extension of- the *GenerateGoalsStrategy* class (see Figure 2) that uses the *generateGoals* method to create the goals for the firemen agents based on their plans. The objectives generated for the simulation were: (i) to save civilians in hazardous areas; (ii) not to put civilian lives at risk; and (iii) to receive aircraft support. The *generateNorms* method of the *GenerateNormsStrategy* class was used to create norms in the environment aiming only at firemen agents, namely a specific set of agents provided by the *PerceptGenerateNorms* class in the JSAN framework. With the ability to specify how to move the firemen agents, the *MovementStrategy* class has been extended to implement the method to specify the specific form of the firemen agents' movement. For this purpose, we used the *ReactiveMovement* class, which extends the *MovementStrategy* class. The movement strategy of the firemen agents checks whether there is any civilian at risk. If there is, they must make the rescue, and they place their lives at risk. To manage that risk a fireman agent can ask for help that will be sent in the form of land vehicles, aircraft, or even fire.

In the *Norm* class (see Table 1), the attributes were defined so that they can create the sanctions, activation and expiration conditions, which are elements that will be regulated by the norm, and also the norm's deontic concept (see Section II). The *EnvironmentSimulation* class is also responsible for managing a set of strategies for visualizing the simulation information represented by the *ReportStrategy*

class. The social contribution strategy provided by the JSAN framework was used to check the social contribution of a norm in case the norm is fulfilled or violated by a firemen agent. The norm-related information is also tested, that is, (i) the rewards and punishments linked to norms; (ii) the deontic concept associated with norms; and (iii) the plans adopted to comply with the norm.

```

Beginning of the simulation
-----
[commanderAgent] Reported that the weather is bad
[commanderAgent] Reported that workers are in a dangerous place
[fireManAgent] Received information from workers at risk:hazardousLocation

```

Figure 6. Flow of start of simulation of the JSAN.

In Figure 6, the Fireman Commander agent receives a message about bad weather and then receives a second message about some civilians in a hazardous location. After receiving the latter message, the Fireman Commander agent sends an alert to the firemen agents saying that information about the existence of civilians in a hazardous area was received.

```

All Plans
-----
Plan: 1 evacuateWorkers(A)
Plan: 2 useHelicopters(A)
Plan: 3 useTroops(A)
Plan: 4 getMoreTroops(MT)
Plan: 5 getMoreHelicopters(MH)
Plan: 6 getLandBasedHelicopters(LBH)
Plan: 7 returnLandBasedHelicopters(RLBH)
Plan: 8 returnHelicopters(RH)
Plan: 9 returnTroops(RT)

```

Figure 7. Emergency plans described in the simulation.

In this case, the rescue operation plans involve the use of aircraft, ground vehicles, and these plans include asking if the firemen agents need the air or ground support (see Figure 7). These plans are related to the goal: “protect the lives of civilians in hazardous areas.”

```

Instantiated Norm :
-----
Deontic Concept: obligation
action : useHelicopters(A)
Rewards: getMoreTroops(X)
Rewards: getMoreHelicopters(Y)

Punishments:
Deontic Concepts: obligation
action : returnTroops(X)

Norm Reward: getMoreTroops(X)
Reward Norm Contribution For Fulfilling +1: 1
Norm Reward: getMoreHelicopters(Y)
Reward Norm Contribution For Fulfilling +1: 2

```

Figure 8. Norm created in the simulation.

As part of the norm instantiation, (i) the deontic concept in this case is an obligation; (ii) a reward is granted if the norm is met and the agent gets air or ground support, and if the norm is violated the agent will not get ground support for the rescue; (iii) the norm is enabled if there is any person at risk and the norm is disabled when all civilians are safe; and (iv) the element that the norm regulates is the action of using aircraft. The norm Rewards in Figure 8 are the rewards associated with norm and Reward Norm shows how the agent will get the reward if the agent complies with the norm.

Finally Figure 9 illustrates the specific plans the firemen agents decide to use after the norm has been activated because of the existence of civilians in hazardous areas risk.

```
Plans in Select Option
-----
Activation :evacuateWorkers(A):
-----
If: useHelicopters(A)
   Contribution: contribution(-1)
-----
If: useTroops(A)
   Contribution: contribution(1)
-----
Result
-----
{fireManAgent} FireManAgent evacuate using Troops
{fireManAgent} FireManAgent uses troops
```

Figure 9. Firemen agents dealing with active norms in the usage scenario.

If they choose to use aerial vehicles, this will contribute negatively, and if they choose to use land vehicles, their contribution will be positive. Because in the simulation firemen agents use the Selfish strategy to deal with the norms, they decided to evacuate civilians who were in this hazardous area using ground vehicles.

VII. CONCLUSION AND FUTURE WORK

This paper proposes the JSAN framework, a framework for Normative Agent Java Simulation, to build goal-oriented agents that can reason about norms. The implementation helps agents (i) to check if they should adopt or violate the norm; (ii) to evaluate the effects of the fulfillment or violation of the norm on their desires and intentions; and (iv) to select desires and plans according to their choices of fulfilling or violating a norm. The applicability of such an implementation can be verified by using the scenario presented in Section VI, where agents are responsible for planning the evacuation of people that are in hazardous locations [2]. The agents were able to reason about the norms they would like to fulfill, and to select plans meeting an agent's intention of fulfilling or violating the norms.

For future work we are defining an experimental study in order to complete the evaluation of our approach. It is also our aim to study other BDI architectures and platforms in order to investigate the possibility of extending them to support the development of BDI normative agents. We also plan to implement new mechanisms to deal with different levels of agent autonomy and show how different levels of restriction

and communities can influence the satisfaction of a norm application [8, 17]. In the current version of the framework the autonomy-related restriction levels were not taken into account. However, the framework can be enhanced with different levels of restrictions, thus offering the possibility to achieve better results in promoting a desirable social order and, as a result, agents can work in function of the common goals of the society in which they are inserted.

REFERENCES

- [1] Bordini, R. H.; Hübner, J. F.; Wooldridge, M. Programming Multi-Agent Systems in AgentSpeak using Jason. [S.l.]: [s.n.], 2007.
- [2] Cerqueira, S. L. R. et al. Plataforma GeoRisc Engenharia da Computação Aplicada à Análise de Riscos Geo-ambientais. PUC-RIO. Rio de Janeiro, 2009.
- [3] Criado, N., Argente, E., Noriega, P., and Botti, V. Towards a Normative BDI Architecture for Norm Compliance. COIN@ MALLOW2010, pages 65–81, 2010.
- [4] Garcia-Camino, A., Rodriguez-Aguilar, J., Sierra, C., Vasconcelos, W.: Norm-oriented programming of electronic institutions. In: AAMAS, 2006.
- [5] Jadexhomepage, <http://jade.tilab.com/>.
- [6] Jennings, N.; Wooldridge, M. Software Agents, IEEE Review., p. 17-20, 1996.
- [7] Kollingbaum, M.: Norm-Governed Practical Reasoning Agents. PhD thesis, University of Aberdeen, 2005.
- [8] Lopez, F. L.; Luck, M.; D'Inverno, M. Constraining Autonomy through Norms, AAMAS, 2002.
- [9] Lopez, Fabiola López. Social Power and Norms. Diss. University of Southampton, 2003.
- [10] Lopez, L. F.; Marquez, A. A. An Architecture for Autonomous Normative Agents, IEEE, Puebla, México, 2004.
- [11] Meneguzzi, F., Luck, M.: Norm-based behaviour modification in bdi agents. In: Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems, 2009.
- [12] Machado, R. Bordini, R. H. "Running AgentSpeak(L) agents on SIM AGENT", August 1–3, 2002.
- [13] Oren, N., Luck, M., Norman, T.: Argumentation for normative reasoning. In: Proc. Symp. Behaviour Regulation in Multi-Agent Systems, pp. 55–60, 2008.
- [14] Rao, A.S., Georgeff, M.P.: Modeling rational agents within a bdi-architecture. In: Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, 1991.
- [15] Rao, A.S.: Agentspeak(l): Bdi agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038. Springer, Heidelberg, 1996.
- [16] Santos Neto, B. F. D.; Lucena, C. J. P. D. JAFF: implementando agentes auto-adaptativos orientados a serviços, Pontificia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2010.
- [17] Santos Neto, A deontic Approach for the Development of Autonomous Agents Normative. Pontifical Catholic University - PUC-Rio - Rio de Janeiro, RJ - Brazil, 2012.
- [18] Silva, V. T. D.; Lucena, C. J. P. D. Modeling Multi-agent Systems, Communications of ACM, p. 103-108, 2007.
- [19] Silva, V.: From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. In: JAAMAS, pp. 113–155, 2008.
- [20] Viana, M. L., Cunha, F. P., Santos Neto, B., Alencar, P., Lucena, C. J. P. A Framework for Supporting Simulation with Normative Agents. WESAAC, 2015. (To Appear).
- [21] Wooldridge, M. and Jennings, "N. R. Pitfalls of agent-oriented development," Proceedings of the Second International Conference on Autonomous Agents (Agents'98), ACM Press, pp. 385-391, 1998.

Quantitative Reasoning of Goal Satisfaction in the i^* framework

Chitra M Subramanian, Aneesh Krishna, Raj P. Gopalan

Department of Computing, Curtin University
Perth, Western Australia
chitra.muniyapp@postgrad.curtin.edu.au, {A.Krishna,
R.Gopalan}@curtin.edu.au

Arshinder Kaur

Department of Management Studies, Indian Institute of
Technology Madras, Tamil Nadu, India
and School of Information Systems, Curtin Business
School, Curtin University, Australia
arshinder@iitm.ac.in, Arshinder.kaur@curtin.edu.au

Abstract—In requirement analysis, goal models play an important role in assessing alternative design options of a software system. Many qualitative and quantitative goal reasoning approaches have been proposed for goal models such as Knowledge Acquisition in Automated Space (KAOS), Non-Functional Requirements (NFR), and Goal Oriented Requirement Language (GRL). However, for i^* goal model only qualitative reasoning has been proposed in Requirement Engineering literature. The aim of this paper is to present a quantitative goal reasoning for i^* goal model. The proposed approach was validated with case studies from existing literature and offers a guide in the decision process. To support the validation a simulation was developed in Visual C++.

Keywords- Requirements engineering, i^* goal model, quantitative analysis, fuzzy numbers

I. INTRODUCTION

In the early stages of Requirement Engineering (RE), goal models are considered a convenient way for modeling and reasoning about alternative design solutions of any software system [1-4]. These models through refinement hierarchy denote the alternative ways of achieving the stakeholder's goals. These different alternative designs have disparate impact on the system goals and different degrees of goal satisfaction. Alternative design solutions are assessed based on some evaluation criteria to choose the best among them. Many qualitative and quantitative reasoning approaches have been proposed in RE literature to support the goal analysis [4-7].

Qualitative reasoning is of limited use of reaching conclusions as the labels become ambiguous in the propagation algorithm. Also, it provides only a quick approach for coarse evaluation in the early stages of the RE process. Quantitative reasoning leads to limited conclusions due to the lack of accuracy and measurability of goal formulations. It is also crucial to assign definite numbers to stakeholder's requirements as requirement elicitation may involve distinct stakeholders having different preferences for the same requirements. The rationale behind it is that distinct stakeholders have different levels of knowledge, training and skills [8]. Moreover in reality, linguistic terms such as low cost, high profit are generally used by the stakeholders to communicate their requirement preferences. The linguistic representation of stakeholder's requirement preferences can be more easily expressed in Fuzzy Logic [9].

The i^* frameworks are useful in qualitatively representing and analysing how stakeholders goals influence each other [7]. However, the existing RE literature seems to be lacking for some method of quantitative support for goal analysis. The objective of this paper is to present a fuzzy based quantitative analysis for evaluating different design alternatives and to identify the best one among them.

The remainder of the paper is structured as follows: Section 2 proposes the fuzzy based quantitative analysis for i^* goal models; Section 3 presents simulation and validation of our work; Section 4 discusses related works; Section 5 concludes the paper.

II. THE QUANTITATIVE FUZZY BASED REASONING OF GOALS

The proposed approach involves the selection of an option based on the impact of the alternative options on soft goals. The results are examined from the point of view of each actor. The goal impacts represented by make, help, hurt, and break are represented by triangular fuzzy numbers. These impacts along with the soft goal preferences are propagated to the high level soft goals in the goal model to find the level of satisfaction. The option that gives the highest level of satisfaction is selected. For understandability, the Youth Counselling case study as shown in Figure 1 has been used throughout the following section to describe the proposed approach. Due to space restrictions readers are directed to Yu [7, 10] for details on Youth Counselling case study and i^* goal models and Gani [11] for details on fuzzy numbers.

i) **Identify the correlation between goals and soft goals in terms of fuzzy weights:** The correlation between alternative options and leaf soft goals are assigned fuzzy weights and our representation is shown in TABLE 1. This contribution is referred to as \bar{C}_{A^*L} , where A is the alternative option and L is the leaf soft goal. It shows the extent to which an alternative option satisfies a leaf soft goal.

Youth Counselling: For the actor *Kids and Youth*, correlation weights between the alternative option *Use Text Messaging* and the leaf soft goals *Comfortableness with service*, *Anonymity[service]* and *Immediacy[service]* are assigned (0.48, 0.64, 0.80), (0.48, 0.64, 0.8), and (0,0.16, 0.32) respectively (TABLE 2). The correlation links between the other option *Use Cyber Cafe/Portal/Chat Room* and the leaf

TABLE 1. FUZZY VALUES FOR GOAL AND LEAF SOFT GOAL CORRELATION

Name	Fuzzy contribution
Make	(0.64, 0.80, 1)
Help	(0.48, 0.64, 0.80)
Some+	(0.32, 0.48, 0.64)
Some-	(0.16, 0.32, 0.48)
Hurt	(0, 0.16, 0.32)
Break	(0, 0, 0.16)

soft goals are assigned weights (0.48, 0.64, 0.80), (0, 0.16, 0.32) and (0.64, 0.80, 1).

ii) **Assign weights to the leaf soft goals:** The leaf soft goals are assigned weights in percentage from 0 to 100 based on their relative importance. The weight is referred to as ω_L .

Youth Counselling: The leaf soft goals (LSG) *Comfortableness with service*, *Anonymity (service)* and *Immediacy (service)* of the actor *Kids and Youth* are assigned weights based on their relative importance as 50%, 30%, and 70% respectively (TABLE 2).

iii) **Calculation of the leaf soft goal score:** For each alternative, the leaf soft goals are associated with a score showing its satisfaction level. The leaf soft goal score is represented by $\bar{S}_{L(A)}$ and is computed by the equation 1 below:

$$\bar{S}_{L(A)} = \bar{C}_{A*L} * \omega_L \quad (1)$$

Youth Counselling: For the actor *Kids and Youth*, the score for the leaf soft goal *Comfortableness with service* for *Text Messaging* is calculated as below

$$\begin{aligned} \bar{S}_{\text{Comfortablenesswithservice(Text Messaging)}} \\ = (0.48, 0.64, 0.80) * 0.5 = (0.24, 0.32, 0.4) \end{aligned}$$

$$\begin{aligned} \bar{S}_{\text{Comfortablenesswithservice(CyberCafe/Portal/Chat Room)}} \\ = (0.48, 0.64, 0.80) * 0.5 = (0.24, 0.32, 0.4) \end{aligned}$$

TABLE 2 shows the scores of other leaf soft goals.

iv) **Propagation of leaf soft goal scores to find the scores of soft goal:** Once leaf soft goal scores are calculated for each alternative, the scores are propagated backwards to find the scores of the high level soft goals. Soft goals are recipients of multiple contribution links, which can be considered as children of each soft goal. The score is calculated in two steps. In first step, the score of its children are multiplied with their respective impact links. In second step, the combined effects of all the children are taken by using the fuzzy maximum operation. The soft goal score is referred to as $\bar{S}_{SG(A)}$ and is obtained by equation 2 :

$$\bar{S}_{SG(A)} = \bigwedge_{i=1}^n \{ \bar{C}_{SCi} * \bar{S}_{LCi|SCi} \} \quad (2)$$

Where \bigwedge represents fuzzy maximum operation, \bar{C}_{SCi} is the correlation link between a soft goal and its i^{th} child, $\bar{S}_{LCi|SCi}$ is the score of its i^{th} child and ' n ' is the number of its children.

Youth Counselling: For the actor *Kids and Youth*, the score of soft goal *GetEffectiveHelp* for the alternative option *UseTextMessaging* is

$$\begin{aligned} \bar{S}_{\text{GetEffectiveHelp(Text Messaging)}} = \text{MAX} \{ \text{MAX} \{ (0.24, 0.32, 0.4) * (0.64, 0.80, 1), (0.144, 0.192, 0.24) * (0.64, 0.80, 1) \}, (0, 0.112, 0.224) * (0.64, 0.80, 1) \} \end{aligned}$$

$$\bar{S}_{\text{GetEffectiveHelp (Text Messaging)}} = (0.154, 0.256, 0.4)$$

v) **Selection of an alternative with the highest score:** The scores are propagated backwards until we reach the soft

goals that are top in the hierarchy. These soft goals are called top soft goals. These scores are compared to determine the best alternative implementation design option and there by assist the analyst in decision making. This is done from each actor point of view. To obtain quantifiable result, the scores are defuzzified by applying α -cut operation and using an optimal index λ . The λ indicates the degree of confidence and it can take values $\lambda=0$ for pessimistic index, $\lambda=0.5$ for moderate index and $\lambda=1$ for optimistic index.

Youth Counselling: From TABLE 3 we can see that, for all actors the alternative option *UseCyberCafe/Portal/ChatRoom* has the highest score when compared to alternative option *UseTextMessaging*. The option *UseCyberCafe/Portal/ChatRoom* provides the best satisfaction level and hence is selected for all the actors.

In case of scenario where all the actors have same type of alternatives and if the proposed approach gives different alternatives selected for each actor, then the composite score for each alternative is calculated. It is calculated by the summation of all top soft goals scores for each alternative. It is referred to as \bar{S}_{AO} and is given by equation 3 below:

$$\bar{S}_{AO} = \sum_{i=1}^n \bar{S}_{SGi} \quad (3)$$

where \bar{S}_{SGi} is the i^{th} top soft goal score and ' n ' is the number of top soft goals. The sum is normalised if it falls beyond the membership functions defined for goal contributions. The normalised value is defuzzified and the one with the highest is selected.

III. SIMULATION AND EVALUATION

Validation of quantitative models is a concern as they play vital role in critical decision making. The proposed framework was evaluated based on its ability to assist analyst in decision making. To facilitate this, a simulation was developed in Visual C++ and was tested with test cases taken from Meeting Scheduler System and Youth Counselling System (YCS) from existing literature [2, 7]. The proposed approach was found to be effective in deciding the alternative design options. Furthermore, it avoids ambiguity (when one or more goals lead to same label) that arise in the i^* qualitative approach. Algorithm 1 outlines the steps in alternative option selection. The soft goal score distribution graph for YCS is provided in Figure 2.

There are many qualitative [4, 5, 7] and quantitative approaches [6, 13] for goal analysis in the existing literature. In our approach, we use fuzzy numbers for alternative selection and goal estimations. By using fuzzy numbers, our approach can handle imprecise and vague requirements of stakeholders like high quality, low cost, good performance. On comparing with qualitative approach, our approach also avoids the requirements conflicts in decision making that arise in qualitative analysis. In quantitative analysis the alternative options are selected based on its impact to the leaf soft goals. It does not take into account its impact on other soft goals in goal graph. Our approach selects the alternative option based on the impact of the alternatives on soft goals by propagating the scores of the leaf soft goals to the top soft goals. Hence we can say that the alternative that is selected will have better satisfaction of the top soft goals.

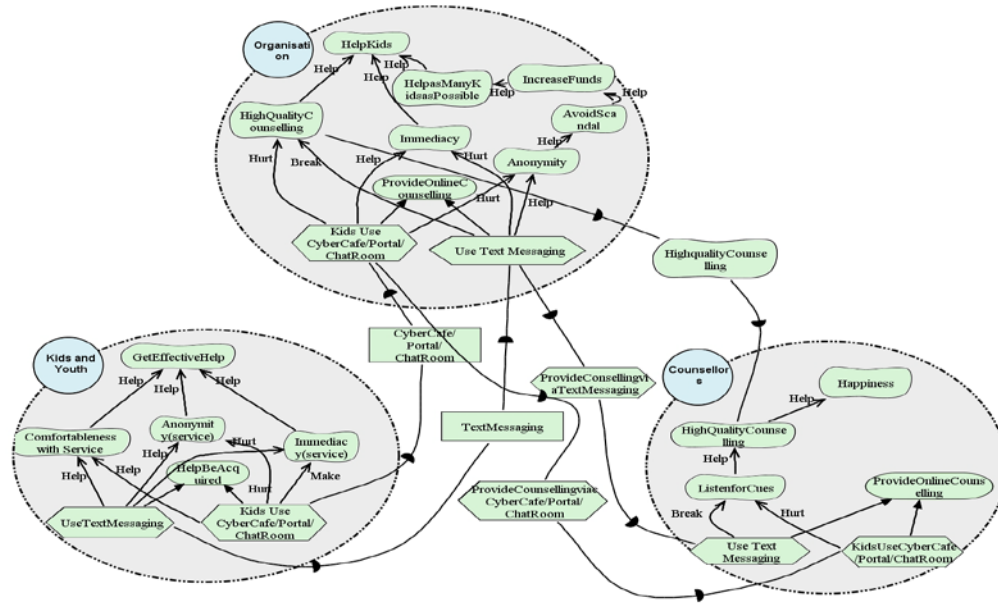


Figure 1. An SR Model: Youth Counselling Example (adapted from [7]).

TABLE 2. LEAF SOFT GOAL SCORES FOR ALL THREE ACTORS

LSG	Weight	Impact		Score	
		Use Text Messaging	Use CyberCafe/Portal/ChatRoom	Use Text Messaging	Use CyberCafe/Portal/ChatRoom
Comfortableness	0.5	(0.48,0.64,0.8)	(0.48,0.64,0.8)	(0.24, 0.32, 0.4)	(0.24, 0.32, 0.4)
Anonymity	0.3	(0.48,0.64,0.8)	(0.0,0.16,0.32)	(0.144, 0.192, 0.24)	(0, 0.048, 0.096)
Immediacy	0.7	(0,0,0.16)	(0.64, 0.80,1)	(0, 0.112, 0.224)	(0.448, 0.56, 0.7)
ListenforCues	0.3	(0,0,0.16)	(0,0,0.16)	(0, 0, 0.048)	(0, 0.048, 0.096)
HighQualityCounselling	0.5	(0,0,0.16)	(0,0,0.16)	(0, 0, 0.08)	(0, 0.08, 0.16)
Immediacy	0.7	(0,0,0.16)	(0.48,0.64,0.8)	(0, 0.112, 0.224)	(0.336, 0.448, 0.56)
Anonymity	0.3	(0.48,0.64,0.8)	(0,0,0.16)	(0.144, 0.192, 0.24)	(0, 0.048, 0.096)

TABLE 3. TOP SOFTGOAL SCORES (*INDICATES GOAL SELECTION)

Actor	Top Soft Goals	Alternative option Score		Defuzzified scores	
		Use Text Messaging Text	Use CyberCafe/Portal/Chat Room	Use Text Messaging Text	Use CyberCafe/Portal/ChatRoom
Kids and Youth	Get Effective Help	(0.154, 0.256, 0.4)	(0.287, 0.448, 0.7)*	0.53(53%)	0.94(94%)
Counsellors	Happiness	(0, 0, 0.03)	(0, 0.02, 0.064)*	0.0075(0%)	0.02(2%)
Organization	Help Kids	(0.004, 0.072, 0.18)	(0.16, 0.29, 0.45)*	0.05(5%)	0.15(15%)

IV. RELATED WORK

Lamsweerde [4] proposed a lightweight quantitative alternative evaluation system for KAOS framework. The proposal uses variables like gauge variable, ideal target value, maximum acceptable value associated with each soft goal. The values of these variables are obtained from the specification of the system. Lamsweerde et al. [12] proposed more accurate, but heavy weight approach based on probability's

interpretation of numbers. Bayesian Networks concepts are used for making predictions about soft goals. This approach becomes difficult to use in case of a complex system.

Affleck et al. [13, 14] proposed a process-orientated, lightweight, quantitative extension to the NFR Framework. The objective of their proposal is to apply a quantitative approach for the decision process and impact of the decision over the system. Liaskos et al. [15] employed the Analytic Hierarchy

Algorithm 1: Goal/task Elicitation (Alternative Selection) and to find the satisfaction levels of intentional elements and actor

```

for each graph in the collection do
  Assign weights to each Leaf Soft Goal (LSG)
  for each task/goal associated in the graph do
    Find task/goal impact associated with each Soft Goal (SG)
    Calculate the LSG score
    for each SG in the graph do
      Calculate the SG score
    end for
  end for
  Defuzzify the top soft goal score
  Select the task/goal with the highest score
end for
case 1: the tasks/goals are same for each input graph
  if the task selected from each graph is different then
    For each task in the input graph
      Add the top soft goal scores of each graph
      Normalize the added score
      Defuzzify and select the highest one
    else
      Compare the alternative scores to select the highest
    end if
  case 2: the tasks/goals are different for each actor in input graph
    Display the task/goal selected for each graph

```

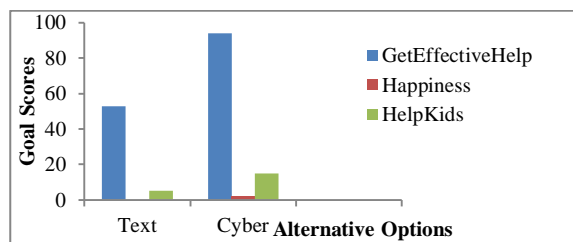


Figure 2. Soft Goals Score distributions

Process (AHP), a mathematical decision making method for goal elicitation in the semi-formal goal models. This suit for goal models that can be viewed as individual AHP problems based on eliciting contribution levels.

Horkoff and Yu [7] have proposed an interesting approach that evaluates goal achievement in enterprise models. However, the main issue with their approach is ambiguity of decision-making process when one or more goal receives the same labels. Furthermore, it requires frequent customer intervention.

Sidiq and Jain [16] proposed a fuzzy based AHP for requirements prioritization. The AHP pairwise comparison are used for assigning weights to goals/soft goals and finds the prioritized list of requirements using binary sort tree method. A. Teka et al. [17] applied fuzzy based reasoning to compare NFR and TROPOS to analyse the impact of goals and requirements changes in goal models. This approach suffers from specifying goal satisfaction levels in terms of concrete numbers.

V. CONCLUSION

This paper proposed a quantitative approach for analysing goals in i^* framework. Compared with the qualitative analysis of i^* framework, our approach strengthens the decision process by avoiding ambiguities and making decisions when the requirements are fuzzy. The proposed approach was validated by applying it to test cases such as Youth Counselling, Meeting Scheduler. In addition, a simulation was developed in Visual C++ to support the evaluation. As a future work, the approach will be extended by including inter-actor dependencies. Furthermore, we plan to apply goal optimisation to i^* goal models.

REFERENCES

- [1] Mylopoulos, J., et al., Exploring alternatives during requirements analysis. *Software, IEEE*, 2001. p. 92-96.
- [2] van Lamsweerde, A., Reasoning about alternative requirements options, in *Conceptual Modeling: Foundations and Applications*. 2009, Springer. p. 380-397.
- [3] Giorgini, P., et al., Reasoning with goal models, in *Conceptual Modeling—ER 2002*. 2003, Springer. p. 167-181.
- [4] Liaskos, S., et al., Representing and reasoning about preferences in requirements engineering. *RE*, 2011. **16**(3): p. 227-249.
- [5] Amyot, D., et al., Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, 2010. **25**(8): p. 841-877.
- [6] Franch, X. On the quantitative analysis of agent-oriented models. in *Advanced Information Systems Engineering*. 2006. Springer.
- [7] Horkoff, J. and E. Yu, Evaluating goal achievement in enterprise modeling—an interactive procedure and experiences, in *The Practice of Enterprise Modeling*. 2009, Springer. p. 145-160.
- [8] Wang, X.-T. and W. Xiong, An integrated linguistic-based group decision-making approach for quality function deployment. *Expert Systems with Applications*, 2011. **38**(12): p. 14428-14438.
- [9] Zadeh, L.A., The concept of a linguistic variable and its application to approximate reasoning—II. *Information sciences*, 1975. **8**(4): p.301-357.
- [10] Eric Yu, *Social modeling for requirement engineering*, 2011: *Mit Press*.
- [11] A.N. Gani: A new Operation on triangular fuzzy numbers for solving fuzzy linear programming problem. *Applied Mathematical Sciences*, Vol. 6, no. 11, 525-532 (2012).
- [12] E. Letier and A. van Lamsweerde, Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the 12th International Symposium on the Foundation of Software Engineering (FSE-04)*. Newport Beach, CA, pp. 53– 62 (2004).
- [13] A. Affleck, and A. Krishna: Supporting quantitative reasoning of non-functional requirements: A process-oriented approach *Int. Conf. Software and System Process (ICSSP)*, Zurich, Switzerland, June, pp. 88–92. IEEE Computer Press, Los Alamitos, CA, USA (2012).
- [14] A. Affleck, A.Krishna and N.R. Achuthan, Selection of Operationalizations for Non-Functional Requirements. *Proc. 9th APCCM '13*, Vol. 143, pp. 69-78. Australian computer Society, Inc., Australia.
- [15] S. Liaskos, R. Jalman, and J. Aranda : On Eliciting Contribution Measures in Goal Models. *Proc. 20th IEEE Int. Requirements Engineering Conf. (RE'12)*, USA, September, pp. 221–230 (2012).
- [16] Mohd Sidiq and S.K. Jain, Applying fuzzy preference relation for requirements prioritization in goal oriented requirements elicitation process, *Int. J. Syste. Assur. Eng. Manag.* (2014)
- [17] A. Teka, N. Condori-Fernndez, I. Kurtev, D. Quartel, and W. Engelsman, Change Impact Analysis of Indirect Goal Relations: Comparison of NFR and TROPOS Approaches Based on Industrial Case Study, in *MoDRE 2012*, Chicago, USA, 2012, pp. 58–67.

CQV-UML Tool: a tool for managing the impact of change on UML models

Dhikra Kchaou
Mir@cl Laboratory,
University Of Sfax, Sfax, Tunisia
Dhikra.kchaou@fsegs.rnu.tn

Nadia Bouassida
Mir@cl Laboratory,
University Of Sfax, Sfax, Tunisia
Nadia.Bouassida@isimf.rnu.tn

Hanène Ben-Abdallah
King Abdulaziz University,
Jeddah, KSA
hbenabdallah@kau.edu.sa

Abstract— An automated change impact analysis and management approach is vital to handle the complexity of adapting software during its evolution. Such an approach reduces the maintenance cost and provides for adequate decision making when confronted with the choice of accepting or ignoring changes. This paper presents a change impact management approach between UML models. It verifies the consistency and the quality of interdependent diagrams after a change is handled. In addition, it calculates the effort required in managing any change and displays a report indicating to the designer all necessary modifications to keep the design coherent. The approach is supported by a toolset, called CQV-UML tool.

Keywords—change impact; consistency; quality; effort estimation

I. INTRODUCTION

With the continuous evolution of software systems, it comes the need to automate the process of analysing and managing the change impact both on the necessary model updates and quality. Change impact analysis and management of interdependent models while keeping their quality is necessary.

A change impact analysis (CIA) method must firstly identify changes made by the designer. Such precise definition provides for the identification of the change operations and the elements affected directly by this change. Secondly, a CIA method must provide for the needed traceability between the different elements in order to analyse the impact of a change not only in the changed model but also between the interdependent models. Based on the identified change and the traceability between elements, the violated consistencies must be detected using a set of consistency rules. The objective of a change impact management approach should cover not only the detection of inconsistencies, but also the ability to provide for a means to estimate both the effort required to handle a change and its impacts on the quality of the various models. A tool that automates this process while taking into account the quality of the changed models and the effort needed represents a success factor of a change management method.

Several change impact management methods for UML diagrams have been implemented (e.g., [1], [3], [5]). They tried to assist the designer in correcting inconsistencies caused by change. However, these techniques do not evaluate the quality of UML diagrams after evolution. In addition, the effort estimation is not treated.

In this paper, we present the tool support CQV-UML tool to automate the impact of changes on UML models. The tool aims to produce a report with warnings about every change that may deteriorate the quality of the model, recommendations about how to make the design consistent, and the number of corrections required to ensure the consistency of all the models. The designer can use this report to decide about which changes should be rethought and/or canceled. Moreover, assisted by a set of quality rules based on metrics, the designer would have an important support in producing a good quality design, which is an essential determinant of the success of the software project.

The remainder of the paper is organized as follows. Section 2 presents an overview of the existing change impact analysis tools. Section 3 presents our change impact analysis and management approach. Section 4 illustrates the usage of the tool through the T-Rot example. Section 5 summarizes the paper and outlines future work.

II. EXISTING UML CHANGE IMPACT ANALYSIS TOOLS

An idea that all researchers involved in change impact analysis agree on is that a manual change impact analysis is too expensive and error prone and that tool support is necessary. In this section, we briefly present change impact analysis tools that have been proposed in the literature. Essentially, we are interested in tools for change impact management applied to UML models.

For instance, Briand et al., [1] propose the iACMTool which manage the change between class, sequence and state chart diagrams by identifying specific change propagation rules for all types of changes. In order to assist the designer to decide about the change, they propose a measure of distance between a changed element and potentially impacted elements to prioritize the results of impact analysis. However, due to the large number of UML model element types and the large number of change types, the number of impact analysis rules is quite large which makes the process of change impact not easy to implement.

Egyed [3] extends the tool in order to assist the designer in discovering unintentional side effects, locating choices for fixing inconsistencies, and then in changing the design model. In fact, for a given inconsistency and for each element in the scope elements, the tool enumerates a list of choices to correct the change. Since there is a large number of choices, the author tries to reduce this list based on the reason of the inconsistency. However, the number of choices as well as the choices

themselves may be ambiguous for the designer especially with for large diagrams.

Keller et al., [5] present an inconsistency resolution framework implemented as an Eclipse plug-in. The tool takes as input one type of change applied to one or more UML model element and outputs a set of impacted elements. In fact, for a given changed element, the tool implements seven impact analysis rules suitable for seven change types. The tool does not give any support to help the designer to correct the change.

Overall, existing works try to detect inconsistencies caused by changes in order to ensure the consistency within and inter UML diagrams. However, they do not offer any assistance to correct detected inconsistencies and to preserve the quality after the change. In addition, they do not support the generation of the corrected diagrams.

III. THE CHANGE IMPACT ANALYSIS AND MANAGEMENT APPROACH

Our approach allows the identification and measurement of potential side effects resulting from requirement/design changes. Its first originality is that it provides traceability between documented use case, class and sequence diagrams. The second originality of our approach is that, besides verifying the consistency of changed UML diagrams, it verifies their quality using a set of metric based quality rules. The third originality is that it measures the effort needed to manage inconsistencies in order to assist the designer in deciding about which changes should be rethought and/or canceled. The fourth originality is that it is supported by a tool that automates all steps and that automatically generates a new version of the corrected diagrams after resolving the inconsistencies caused by different changes.

A change impact analysis technique needs first of all a way to specify changes. Thus, in a previous work [4], we defined a MOF [2] based change meta-model that covers all change types at a high level of abstraction. Being MOF-based, our change meta-model defines all possible changes affecting the elements independently of a particular modeling language. In addition, it can be extended and adapted to define changes that affect any MOF-based model and, in particular, UML models.

The second hurdle that change impact analysis and management faces is the semantic and structural traceability among the numerous elements of the different diagrams. For this purpose, we propose a graph concept, called "model dependency graph" that encodes the requirements (use case) and design (class, sequence) diagrams in an integrated way. The encoding uses the semantic traceability results and explicitly represents the syntactic relationships among the diagrams' elements. The model dependency graph provides for the needed traceability to analyze systematically the impact of a change on the consistency of the diagrams.

Inspired from the work of Lallchandani et al., [6] for static slicing of UML models, the model dependency graph (MDG) is constructed by transforming the use case, class and sequence diagrams to graphs. In particular, our adaptation accounts for the association relationships (not treated by Lallchandani et al., [6]). In addition, we integrated the documented use case

diagram to the MDG which is also not treated in by Lallchandani et al., [6].

In our approach, the UML class diagram is transformed into a Class Dependency Graph (CDG) and every UML sequence diagram is transformed into a Sequence Dependency Graph (SDG). Every UML use case diagram is transformed into a Use Case Dependency Graph (UCDG) based on a structured use case description [7]. To get all dependencies among the various diagrams, the UCDG, CDGs and SDGs are merged into a Model Dependency Graph (MDG).

To trace the change impact from the use case diagram across the class and sequence diagrams, the CDG and SDGs are firstly integrated into a single graph. The constructed MDG must be completed with the requirements diagram, i.e., the documented use case diagram. For this, we need to identify the correspondence among the ordered actions (specified in the use case scenarios) and the messages in the sequence diagrams. For this purpose, we use an information retrieval technique: term frequency – inverse document frequency (TF-Idf) [8] to measure the cosine similarity measure [12] in order to determine the most resembling message in the sequence diagram to the query (i.e., action in the scenario) and consequently to a use case. In our case, documents and queries contain the set of grammatical units that compose a message/action in SD/UC added to their synonyms extracted from WordNet. The calculus of the different weights for the terms is followed by the calculation of a similarity measure which is the cosine.

After the cosine similarity calculation, the documents (i.e. the actions in the UCs) that are similar to a query (i.e. messages in SD) are linked together in order to construct the MDG. Note that after this step, a validation step may be needed by the designer since the results of the cosine similarity computation may return several ranked possibilities. The designer should validate/select one value that better fits his situation.

Once the change is detected and the traceability is established, the consistency of changed diagrams is verified based on a set of consistency rules. As an example of an inter-diagram consistency rule, each operation in SD must be defined in the receiver's class in CD. A change may affect not only the consistency of UML diagrams, but also their quality. Thus, in addition to the preservation of the consistency of UML diagrams, their quality must be evaluated and preserved after a change is introduced. For instance, the deletion of a class that had many important relationships with other classes, or that participates in a design pattern [10] depreciates the quality of the class diagram. The quality of software can be evaluated using several metrics (cf., [9]) interpreted through a set of thresholds (cf., [11]). In our approach, these metrics and their thresholds are used to evaluate and/or to predict the quality effects of a change in order to propose a set of recommendations to the designer. For this objective, we propose a set of quality rules. To preserve the intra-diagram quality after a change, we calculate the CK metrics suite [9] before and after every change, and based on their thresholds [11], we verify a set of quality rules and we inform the designer about any violation. On the other hand, in order to approximate the effort needed for change impact management, we propose to calculate the number of

required modifications (NRM) necessary when correcting the inconsistencies caused by an intra/inter diagram changes. The NRM sums up the number of update/change operations needed to correct a violated consistency rule.

To calculate NRM, we propose the new intra/inter diagram metrics shown in Table 1. Note that the proposed thresholds for these metrics will be defined using an empirical study in a future work. For a given change, the NRM is the sum of the metrics values, concerned by the change.

TABLE 1: METRICS USED TO MEASURE THE INTRA/INTER-DIAGRAM QUALITY

Metrics		Definitions
Intra-diagram metrics	NAtOp	Number of times an attribute (and operation) of a class is used (called) in an operation.
	NAtPr	Number of times an attribute is used as a parameter in an operation.
	NM2Ob	Number of messages between two objects.
Inter-diagram metrics	NCSd	Number of times a class is used as an object in SDs.
	NATSD	Number of times an attribute is used in SDs.
	NOpSD	Number of times an operation is used as a message in SDs.

IV. CQV-UML TOOL: A TOOL SUPPORT FOR CHANGE MANAGEMENT

To implement our change impact management approach, we have developed a tool named CQV-UML Tool: a Consistency and Quality Verification tool for UML diagrams.

A. Functional architecture

The principal activities performed by our tool are the change detection, the consistency verification and quality verification. The tool takes as input a set of UML models versions corresponding to the original and changed diagrams modelled using the CASE Tool: ARGOUML. The first step transforms the XMI files corresponding to the design diagrams into XML. The transformation is performed thanks to XSLT to obtain reduced representations using the API JDOM. The aim of this step is to eliminate all superfluous information, that is specific to the CASE tool ARGOUML. Secondly, the list of changes is recorded and displayed to the designer. Afterwards, the cosine similarity measure is calculated and our graph based technique (MDG) is implemented in order to establish the traceability between the interdependent diagrams. Based on the achieved traceability, the set of violated consistency and quality rules corresponding to each change type is displayed to the designer. Finally, after accepting the proposed correction, the corrected diagram is generated.

B. CQV-UML tool application

To illustrate the various functionalities of the CQV-UML Tool including the consistency and quality verification of the different diagrams after evolution, let us consider a case study where UML was used to develop a system for autonomous navigation by the intelligent service robot, T-Rot. The use case diagram comprises two use cases (Figure 1): “Navigation” and “Obstacle Avoidance”. The “Navigation” use case textual description is presented in Table 2. The sequence diagram SD corresponding to the Navigation use case is presented in Figure

2. The class diagram CD of this example is presented in Figure 3.

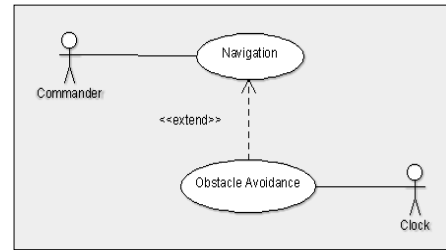


Figure 1. Main use-case diagram of T-Rot system

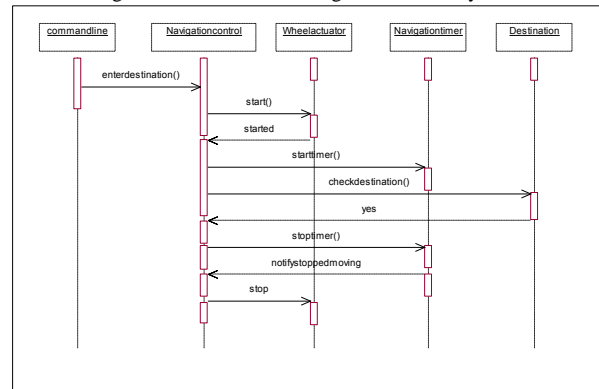


Figure 2. SD1: the sequence diagram of the UC1 “Navigation”

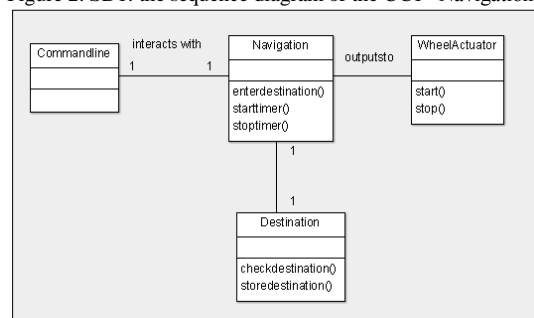


Figure 3. The Robot system class diagram CD

The first step in our approach consists in detecting changes for each UML diagram. The interface 1 of Figure 4 shows the detected changes in the class diagram. By clicking on the change (delete the *Start()* operation from the *WheelActuator* class in the CD), the list of violated consistency rules in the top-right of the screen shot are displayed. In fact, the change violates the consistency rule: Each operation in SD must be defined in the receiver’s class in CD since the deleted *start()* operation exists in the SD1 between the *Navigationcontrol* object and the *WheelActuator* object. This inconsistency is detected thanks to the traceability obtained through the MDG. The CQV-UML tool recommends the designer to correct this inconsistency by deleting the message *start()* in SD1 corresponding to the deleted operation or undo the change. To take the appropriate decision, the CQV-UML tool calculates the number of required modification NRM needed to correct the inconsistency. The NRM is calculated based on the metric NOpSD: number of times the operation is used as a message in SDs. The *start()* operation exist one time in the SD1, so, the designer has one delete to do.

To manage the second change, the CQV-UML tool calculates the similarity measure between the deleted action

NSa3 in the *Navigation* use case and the list of messages in the sequence diagram corresponding to the *Navigation* use case. The interface 3 of Figure 4 presents the similarity results which show that the *NSa3* action corresponds to the *notifystopmoving* message in SD1. The system informs the designer that the deleted action exists as a message in SD and as an operation in CD and proposes to delete them.

TABLE2. "NAVIGATION" USE CASE DESCRIPTION (UC1)

Use case	Navigation
Actor	Commander
Precondition	The robot system has the grid map and the current position.
PostCondition	The robot system is at the destination and waiting for the next destination
Extension Point	[obstacles are recognized], use case « Obstacle Avoidance »
Normal Scenario NS	<p><NSa1><The user ><enters a destination ></p> <p><NSa2><The system> < commands the wheel actuator to start moving to the destination ></p> <p>< NSa3>< The wheel actuator > < notifies the system that it has started moving ></p> <p>< NSa4>< The system> <determines that it arrives at the destination></p> <p>< NSa5>< The wheel actuator><notifies the system that it has stopped moving ></p>
Alternatives Scenario AS	<p>< If the system doesn't arrive at the destination ></p> <p><AS1a1> <the system> < keeps moving ></p>

The quality verification in the interface 2 of Figure 4 shows that the deletion *outputsto* association from CD violates a quality rule. In fact, the *WheelActuator* class becomes isolated. The tool recommends to the designer to add an association or to cancel the change.

When the designer accepts the proposed corrections, the affected diagrams are modified and displayed to the designer. The interface 4 of Figure 4 shows the generation of the corrected class diagram.

V. CONCLUSION

This paper introduced an approach for change impact management and its associated CQV-UML tool. The proposed approach consists in managing the impact of changes affecting elements in UML diagrams essentially use case, class and sequence diagrams. The MDG graph is used to model the inter-

dependencies between the different diagram elements and as a consequence to trace the impact of the change.

We are currently examining how to evaluate the proposed approach on open-source systems and comparing the results of our experiments with other existing approaches.

REFERENCES

- [1] L. C. Briand, Labiche, Y. O'Sullivan, L. Impact Analysis and Change Management of UML Models!. In Proceedings of the International Conference on Software Maintenance, 2003, pp. 276-280.
- [2] OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1, OMG Document Number: formal/2011-08-07, <http://www.omg.org/spec/MOF/2.4.1/PDF>.
- [3] A. Egyed, Fixing Inconsistencies in UML Design Models. Proceedings of the 29th International Conference on Software Engineering, 2007, pp. 292-301.
- [4] D. Kchaou, N. Bouassida, H. Ben-Abdallah, A MOF-based change meta-model", Proceedings of the International Arab Conference on Information Technology, CCIS, Zarqa, Jordan, 2012.
- [5] A. Keller, S. Demeyer, Change Impact Analysis for UML Model Maintenance!. Book chapter: Emerging Technologies for the Evolution and Maintenance of Software Models, 2012, pp. 32-56.
- [6] J.T. Lallchandani, R. Mall, Static Slicing of UML Architectural Models, Journal of object technology, Vol. 8, No. 1, 2009, pp. 159-188.
- [7] M. Ali, H. Ben-Abdallah, F. Gargouri, Towards a Validation Approach of UP Conceptual Models, In : Proceeding of Consistency in Model Driven Engineering in European Conference on Model Driven Architecture - Foundations and Applications Nuremberg, Germany, 2005, pp. 143-154.
- [8] H. Wu and R. Luk and K. Wong and K. Kwok. "Interpreting TF-IDF term weights as making relevance decisions". ACM Transactions on Information Systems, 26 (3). 2008.
- [9] S.R. Chidamber, C.F. Kemerer, Towards a metrics suite for object oriented design. In Conference proceedings of Object-oriented programming systems, languages, and applications, 1991, pp. 197-211.
- [10] Bouassida N., Ben-Abdallah, Issaoui I. Evaluation of an automated multi-phase approach for pattern discovery ", International Journal of Software Engineering and Knowledge Engineering, World Scientific, Vol 23, N10, pp 1367-1398 (2013).
- [11] E. Chandra, P. Linda, Class Break Point Determination Using CK Metrics Thresholds!, Global Journal of Computer Science and Technology, Vol. 10, 14, 2010, pp. 73-77.
- [12] A. Singhal, Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24, 2013, pp. 35-43.

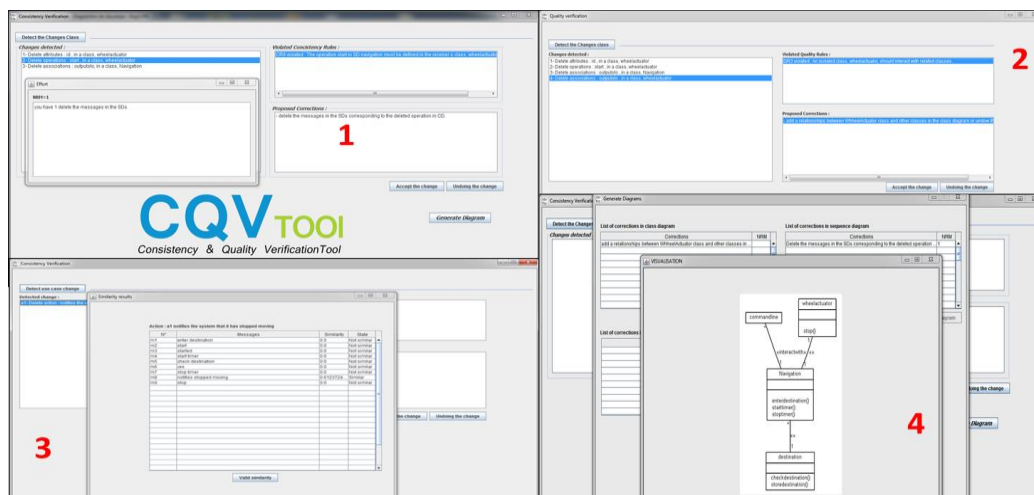


Figure 4: Snapshots of the CQVUML tool

An evolution management model for multi-level component-based software architectures

Abderrahman Mokni¹, Marianne Huchard², Christelle Urtado¹, Sylvain Vauttier¹, and Yulin Zhang³

¹LGI2P / Ecole des Mines d'Alès, Nîmes, France, {Abderrahman.Mokni, Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

²LIRMM / CNRS & Montpellier University, France, huchard@lirmm.fr

³Laboratoire MIS, Université de Picardie Jules Verne, Amiens, France, yulin.zhang@u-picardie.fr

Abstract

Handling evolution in component-based software architectures is a non trivial task. Indeed, a series of changes applied on software may alter its architecture leading to several inconsistencies. In turn, architecture inconsistencies lead to software erosion and shorten its lifetime. To avoid architectural inconsistencies and increase software reliability, architecture evolution must be handled at all steps of the software lifecycle. Moreover, changes must be treated as first class entities. In this paper, we propose an evolution management model that takes these criteria into account. The model is a support for our three-level Dedal architectural model. It captures and handles change at any of the Dedal abstraction levels: specification, implementation and deployment. It generates evolution plans using evolution rules proposed in previous work. The generation process is implemented using the ProB model checker and evaluated through three evolution scenarios of a Home Automation Software.

Keywords: software architecture evolution, component reuse, evolution rules, evolution management, abstraction level, change propagation, consistency checking.

1 Introduction

Software evolution [1] is becoming more and more challenging due to the increasing complexity of software systems and their importance in everyday life. While component reuse has become crucial to shorten large-scale soft-

ware systems development time, handling evolution in such systems is a serious issue. As witnessed by Garlan *et al.* in their recent study [2], the difficulty of reuse lies essentially on architectural mismatches that arise due to several changes that affect software. A famous problem is software architecture erosion [3, 4]. It arises when modifications of the implementation of a software violate the design principles captured by its specification architecture. Such erosion leads to software degradation and shortens its lifetime. Increasing confidence in reuse-centered, component-based software systems lies on resorting out multiple issues.

First, software architectures must support change at any step of component-based development to meet new user needs, improve component quality, or cope with component failure. Second, the impact of change must be handled locally (at the same abstraction level) to avoid architecture inconsistencies and propagated to the other abstraction levels to avoid incoherence between the architecture descriptions, notably erosion. Third, the evolution activity must be tracked to enable monitoring, commitment and/or rollback and versioning. In this paper, we propose an evolution management model that deals with all these issues. It is based on our Dedal architectural model [5, 6] that covers the three main steps of component-based software development: specification, implementation and deployment. The model uses architecture properties and evolution rules proposed in previous work [7, 8] to generate evolution plans that preserve the consistency of all architecture descriptions as well as coherence between them. The remainder of this paper is outlined as follow: Section 2 gives the background of this work. Section 3 presents the evolution management model. Section 4 presents the evolution plan generation process, an implementation and evaluation. Section 5 discusses

related work before Section 6 concludes the paper and gives future work directions.

2 Background

This article is concerned with evolution management in multi-level component-based architectures. Specifically, we address software architectures described by Dedal [6], a three-level architectural model. First, we introduce Dedal and then we give a brief overview of its formalization.

2.1 Dedal a three-level architectural model

Dedal is a novel architectural model that covers the three main steps of component-based development by reuse: specification, implementation and deployment. The idea of Dedal is to build a concrete software architecture (called configuration) from suitable software components stored in indexed repositories. Candidate components are selected according to an intended architecture (called specification) that represents an abstract and ideal view of the software. The implemented architecture can then be instantiated (the instantiation is called assembly) and deployed in multiple contexts.

The Dedal model is then constituted of three descriptions that correspond to three architecture abstraction levels.

The architecture specification corresponds to the highest abstraction level. It is composed of component roles and their connections. Component roles encapsulate the required functionalities of the future software.

The architecture configuration corresponds to the second abstraction level. It is composed of concrete component classes selected from repositories and realize the identified component roles in the architecture specification.

The architecture assembly corresponds to the lowest abstraction level. It is composed of component instances that instantiate the component classes of the architecture configuration. While the specification and configuration descriptions represent the software at design-time, the assembly description represents the software at runtime.

Figure 1 illustrates the three abstraction levels of Dedal on an example of a Home Automation Software. The software enables to manage the light of buildings in function of the time through an orchestrator (component role *HomeOrchestrator*). The specified functionalities are turning on/off the light (component role *Light*), controlling its intensity (component role *Intensity*) and getting information about the time (component role *Time*).

2.2 Dedal formalization

Dedal formalization is crucial to enable the verification and validation of the derived architectural models as well

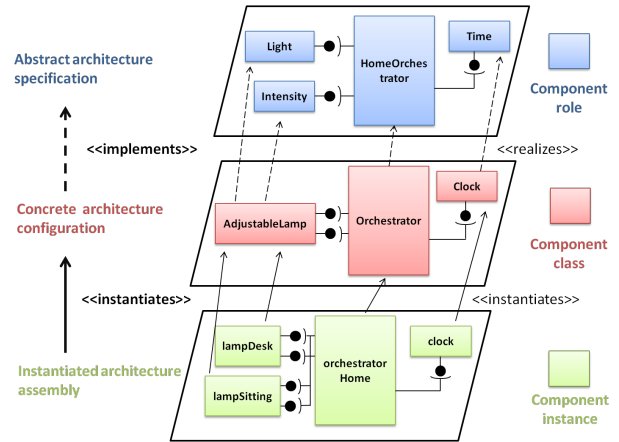


Figure 1. Example of a Dedal model: Home Automation Software

as evolution management. In [7], we propose a formalization of Dedal that comprises two kinds of typing rules. The first kind is about intra-level rules. These rules define the relations between components of the same abstraction level such as compatibility and substitutability. The second kind is about inter-level rules. These rules define the relations between components of different abstraction levels (*cf.* Figure 1 for inter-level relations). For instance, the realization rule checks whether a (or a set of) component class(es) realizes a (or a set of) component role(s). This rule is also used to search for candidate concrete components in repositories to implement a specified software architecture. Inter-level and intra-level rules are generalized to support the architectural level. First, using intra-level rules, we can check architecture consistency at any abstraction level. Second, using inter-level rules, it is possible to check coherence between architecture descriptions at different abstraction levels. For instance, we can check whether a configuration implements all the desired functionalities documented in its specification. In [8], we propose a set of evolution rules that enable to evolve Dedal models. The proposed rules allow the manipulation (addition, deletion and substitution) of architectural elements (*e.g.*, components, connections) at the three Dedal abstraction levels.

In this work, we present an evolution management model for architecture models derived from Dedal. We show how evolution rules can be used to generate evolution plans that preserve the consistency of architecture descriptions and coherence between them.

3 The evolution management model

Figure 2 presents our evolution management model. It is composed of three parts: the architectural Model (meta-

classes of group 1), the changes that affect the architectural model (meta-classes of group 2) and the evolution manager (meta-classes of group 3).

3.1 The architectural model

The architectural model is the target of change. It is derived from the Dedal meta-model and hence includes three architecture descriptions, each represents the component-based software at a different abstraction level.

Architecture properties represent the set of rules that decide about the well-formedness of the architectural model. These properties have to be preserved after change and hence are part of the analysis goals that must be enforced during the software evolution. In our work we focus on two properties,

Architecture consistency states whether the elements of the architecture are correctly typed and well connected (each interface is connected to a compatible one). Additionally, consistency involves the internal completeness of the architecture, *i.e.*, an architecture is said complete if all its required properties are met. From a structural viewpoint, an architecture is complete if all its required interfaces are connected to compatible provided ones.

Architecture coherence states whether an architecture description at an abstraction level is in conformance with an architecture description at an adjacent abstraction level. Verifying architecture coherence keeps all architecture descriptions up-to-date and avoids the problems of drift and erosion.

Model manipulation operations are elementary change operations that manipulate the artifacts of the architectural model (*e.g.*, component roles, component classes, or connections). They are classified into three types: addition, deletion and substitution. Manipulation operations are composed of four parts. A signature defines the operation name and states its arguments. Preconditions are related to the architectural model (*e.g.*, a precondition checks if substitutability between two components is possible). Actions are applied on the architectural model by updating the set of its artifacts. Post-conditions must be verified by the new state of the architectural model after applying the actions of the operation.

3.2 The architectural change

Architectural changes characterize all the modifications that alter the software architecture. They meet new requirements to keep software up-to-date or arise due to an environmental change (*e.g.*, lack of resources, or faults). Software architectures are subject to change at any abstraction level of component-based development. The impact of change may affect the other abstraction levels. All of these

facts about software change make its handling a non trivial task. In order to tackle this complexity, we represent change as a first class entity that interferes with the architectural model. Furthermore, we identify three main change characteristics necessary for the evolution management process: origin, level and subject.

There are two types of **change origin**: *initiated change* and *triggered change*. Initiated change has an external source. It may be originated from user action or from the execution environment. Triggered change is internal and is induced by the evolution manager to reestablish the architecture consistency at the same abstraction level (*local change*) and/or preserve conformance between all architecture descriptions at other abstraction levels (*propagated change*).

Change level designates the level where a change is currently performed. It allows to identify which properties must be checked to ensure a successful evolution.

Change subject designates the artifacts subject to change. This information is useful to identify the elements that have to be manipulated in the evolution process.

3.3 The evolution manager

The evolution manager captures software change and controls its impact on the architectural model. It uses *evolution rules* to generate an *evolution plan* that satisfies a given *evolution goal*. The evolution manager ensures also the co-evolution of the descriptions of the software architectures at the other abstraction levels.

Evolution rules are specific operations that are composed of model manipulation operations. They manage and control access to these operations using preconditions related to the change, according its origin, level, or subject.

The **evolution goal** sets all the conditions that must be satisfied by the architectural model after the change. Basically, it is composed of two parts: architecture properties (*i.e.*, consistency and coherence) and the post-condition of the initiated change.

4 The generation process

In this section, we state the generation process problem. Then, we test and evaluate two search strategies implemented by the ProB [9] model-checker.

4.1 Problem formalization

Notations: Let M be an instance of the Dedal architectural model and S be the state space of M . Then, let L be the enumeration of Dedal abstraction levels. $L = \{specLevel, configLevel, asmLevel\}$. Let E be the set of all evolution rules and E_l the subset of E related to an abstraction level l , $l \in L$. For each $e_i \in E$, let $pre(e_i)$ be

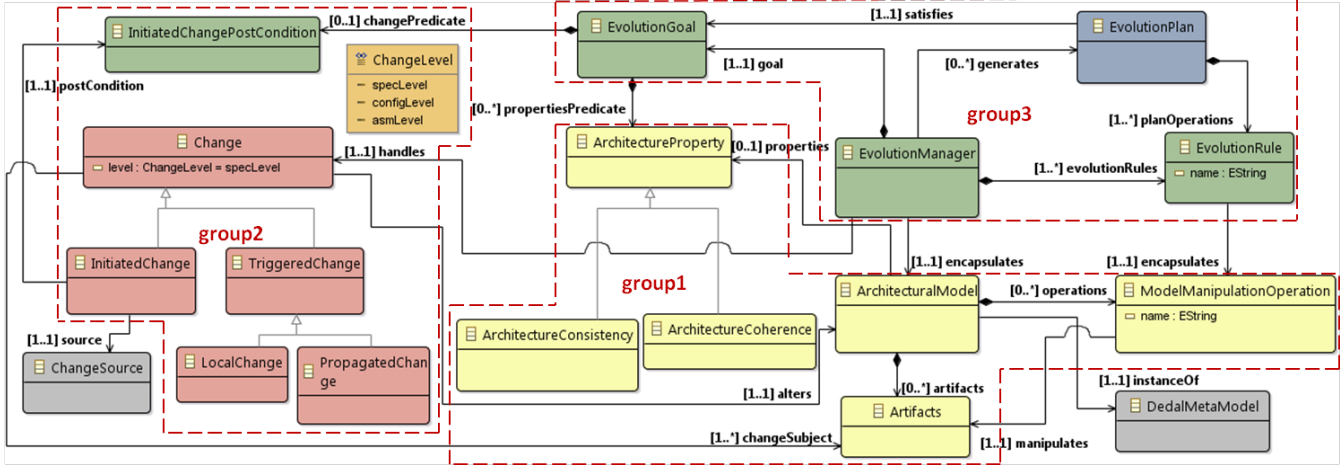


Figure 2. The evolution management model (ecore)

the precondition of e_i . Let C_{i_l} be the change initiated at an abstraction level l and $post(C_{i_l})$ the post-condition satisfied by $s \in S$ after applying C_{i_l} on M . Let $P_{consistency}(l)$ be the consistency property related to the abstraction level l and $P_{coherence}(l, k)$ be the coherence property between the two adjacent abstraction levels l and k . Finally, let G_l be an evolution goal related to an abstraction level l .

Problem: Considering an initiated change C_{i_l} on M , we would like to (1) find a sequence of $e_i \in E_l$ where $G_l = post(C_{i_l}) \wedge P_{consistency}(l)$ is satisfied and (2) $\forall l, k$ where $P_{coherence}(l, k) = false$, find a sequence of $e_k \in E_k$ where $G_k = P_{coherence}(l, k) \wedge P_{consistency}(k)$ is satisfied. The evolution plan Pl is thus the concatenation of all found sequences. $Pl = Pl_{local}; Pl_{propagated}$ where Pl_{local} is the plan related to the local change and $Pl_{propagated}$ is the concatenation of the plans related to the propagated change.

4.2 Implementation overview

The implementation is composed of two parts: the Dedal modeler and the ProB [9] model-checker. The Dedal modeler is an eclipse-based tool that enables the creation and edition of Dedal diagrams (*i.e.*, specification, configuration and assembly diagrams) and the automatic generation of B [10] formal models corresponding to those diagrams. ProB is a model-checker and animator for B models. It calculates and simulates state-transitions. In our case, it can calculate all the enabled evolution rules (defined as B operations) at each state of the model. Moreover, using forward chaining inferences, ProB can search for a sequence of evolution rules that reaches the evolution goal. It proposes depth-first (DF), breadth-first (BF) and mixed depth-first/breadth-first (DF/BF) search strategies to find invariant violations or defined goals. The most suitable strategy de-

pends on the kind of checking the user wants to perform. In the case of searching for a specific state by exploring a large state space, depth first seems to be the most efficient strategy as stated in [11]. In the remainder, we use ProB to generate evolution plans and observe the resolution time using DF and mixed DF/BF. We compare the results to see which strategy is the most efficient.

4.3 Evaluation

To illustrate the generation process, we run three evolution scenarios on the example of HAS. The initial architectures are illustrated by Figure 3.

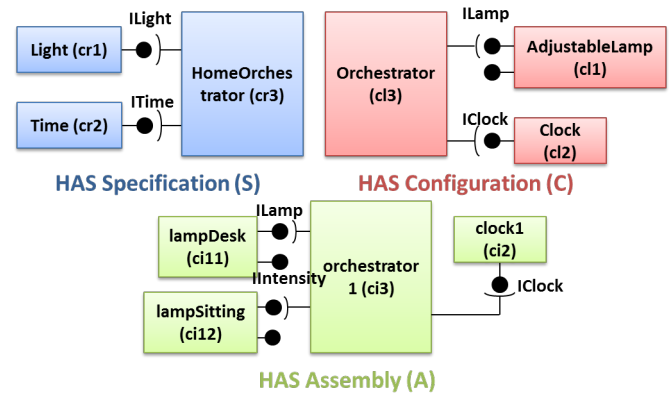


Figure 3. The initial state of HAS

Scenario 1 corresponds to a requirement change. The specification of HAS needs to be evolved to enable the control of the building luminosity. This consists in adding a new role (*cr1a*) that provides a luminosity functionality.

Scenario 2 corresponds to an implementation change.

The configuration needs to be evolved to turn under android OS. The initiated change consists in adding an android orchestrator (*cl3a*). This entails to delete the current orchestrator (*cl3*).

Scenario 3 corresponds to a runtime change. The *clock1* (*ci2*) component instance must be replaced due to a dry battery of the clock. That of the *embeddedClock1* (*ci2a*) mobile device is used instead.

We run two tests for each evolution scenario. The first test uses the DF strategy and the second one uses the DF/BF strategy. The results (*cf.* Table 1) show that DF is more efficient than BF/DF in all cases.

	Change level	DF (seconds)	DF/BF (seconds)
Scenario1 (top-down change)	specLevel (initial)	2.36	48
	configLevel	16.71	4.97
	asmLevel	9.12	23.83
	full process	28.19	76.8
Scenario2 (mixed)	configLevel (initial)	9	13.39
	specLevel	2.98	5.37
	asmLevel	12.01	65.41
	full process	23.99	84.17
Scenario3 (bottom-up change)	asmLevel (initial)	4.5	26.54
	configLevel	77.8	136.08
	specLevel (not affected)	0	0
	full process	82.3	162.62

Table 1. Evaluation results

Due to space limitation, we only show the generated plans corresponding to `scenario2`. Figure 4-a, Figure 4-b and Figure 4-c respectively show the ProB output for local change (configuration level), bottom-up change (specification level) and top-down change (assembly level). The sequence must be read from the bottom to the top of the output view. We use the following notation to explain the syntax of the generated rules: *cr*, *cl*, *ct* and *ci* correspond respectively to component role, class, type and instance. The *pint* and *rint* prefixes respectively denote a provided or a required interface. *HASSpec*, *HASConfig* and *HASAssm* respectively name the HAS specification, configuration and assembly.

5 Related work

Managing software architecture evolution is still an open issue. For more than two decades, a lot of efforts have been dedicated to provide methods, tools and techniques for architecture modeling and analysis. Several ADLs (Architecture Description Languages) [12] have been proposed. Most of them provide textual notations for describing and analyzing software systems. They are usually supported by tools or integrated environments for edition, analysis and simulation. Examples include C2SADL [13], Wright [14], Darwin [15] and ArchJava [16]. C2SADL is an ADL for the design of concurrent systems. It includes a sub-architecture modification language (AML) to support evolution. Changes are first applied at the architectural level and then implemented using a runtime infrastructure. Wright is

```
config_removeClass(HASConfig,cl3)
config_class_disconnect(HASConfig,(ct2|->pintlClock),(ct3|->rintlClock))
config_class_connect((ct3a|->rintlLamp2),(ct1|->pintlLamp2),HASConfig)
config_class_connect((ct3a|->rintlIntensity),(ct1|->pintlIntensity),HASConfig)
config_class_connect((ct3a|->rintlClock2),(ct2|->pintlClock),HASConfig)
config_addServer(HASConfig,(ct1|->pintlIntensity))
config_class_disconnect(HASConfig,(ct1|->pintlLamp2),(ct3|->rintlLamp))
config_addClass(HASConfig,cl3a)
```

a- Local change

```
spec_connect((cr3a|->rintlLight2),(cr1|->pintlLight),HASSpec)
spec_connect((cr3a|->rintlLum),(cr1a|->pintlLum),HASSpec)
spec_connect((cr3a|->rintTime2),(cr2|->pintTime),HASSpec)
spec_addRole(HASSpec,cr3a)
spec_addRole(HASSpec,cr1a)
spec_removeRole(HASSpec,cr3)
spec_disconnect(HASSpec,(cr1|->pintlLight),(cr3|->rintlLight))
spec_disconnect(HASSpec,(cr2|->pintTime),(cr3|->rintTime))
```

b- Bottom-up change

```
asm_bind((ci3a|->rintlLamp2Inst),(ci11|->pintlLamp2Inst2),HASAssm)
asm_bind((ci3a|->rintlIntensityInst),(ci11|->pintlIntensityInst2),HASAssm)
asm_bind((ci3a|->rintlClock2Inst),(ci2|->pintlClockInst),HASAssm)
asm_deployInstance(HASAssm,ci3a,cl3a)
asm_addServerInstance(HASAssm,(ci11|->pintlIntensityInst2))
asm_removeInstance(HASAssm,ci3,cl3)
asm_removeInstance(HASAssm,ci2,cl1)
asm_unbind(HASAssm,(ci11|->pintlLamp2Inst2),(ci3|->rintlLampInst))
asm_deleteServerInstance(HASAssm,(ci12|->pintlLamp2Inst1))
asm_unbind(HASAssm,(ci2|->pintlLamp2Inst1),(ci3|->rintlLampInst))
asm_unbind(HASAssm,(ci2|->pintlClockInst),(ci3|->rintlClockInst))
```

c- Top-down change

Figure 4. generated plans

also a domain-specific ADL. It aids the design of distributed architectures. Wright focuses more on increasing the architect's confidence on the design of systems. It enables consistency checking and analysis using CSP (Communicating Sequential Processes) as a formal basis. However, it does not propose any language or notation to describe architectural changes. Darwin is relatively similar to Wright as it shares the same goal. Unlike Wright, it includes a declarative language that supports change description (including operations such as create, remove, link, or unlink). Beside the fact that they are domain-specific, C2SADL, Wright and Darwin do not support reverse (bottom-up) evolution. Moreover, they cover only the specification level of the software system and hardly support the implementation and deployment levels. This gap between architecture specification and its implementation generates several inconsistencies when applying changes and shortens the software lifetime. ArchJava is an ADL that unifies the architectural level and the implementation code in a single entity. It uses a type system to check conformance between both descriptions. Although, this unification enables co-evolution, it makes it harder to separate program implementation from its specification. Separating specification from implementation is central in our approach to foster component reuse. Moreover, as far as we know, all cited ADL do not handle changes as first class entities, neither do they propose a mechanism for generating evolution plans.

6 Conclusion and future work

This work proposes an evolution management model for component-based software architectures. It represents changes as first class entities as the architecture models derived from Dedal. The model enables to (1) capture change at any architecture abstraction level, (2) handle its impact at the same level and (3) propagate it to the lower and higher abstraction levels keeping then all the architecture descriptions coherent. Both top-down (forward) and bottom-up (reverse) evolution are then supported by the proposed model.

At this stage of work, evolution is not yet automated and is simulated using model-checking techniques on the generated formal models of Dedal [7]. Unfortunately, this is limited by combinatorial explosion. Using meta-heuristic techniques could be a solution to reduce the complexity of evolution plan generation. Ongoing work is the development of an eclipse-based environment of Dedal that automates the evolution plan generation process. Change requests can be expressed using a change description language such as Dedal-CDL [17]. Furthermore, we are considering to set a versioning mechanism for the architectural models derived from Dedal.

References

- [1] T. Mens and S. Demeyer, *Software Evolution*. Springer, 2008.
- [2] D. Garlan, R. Allen, and J. Ockerbloom, “Architectural mismatch: Why reuse is still so hard,” *IEEE Software*, vol. 26, no. 4, pp. 66–69, July 2009.
- [3] D. E. Perry and A. L. Wolf, “Foundations for the study of software architecture,” *SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992.
- [4] L. de Silva and D. Balasubramaniam, “Controlling software architecture erosion: A survey,” *JSS*, vol. 85, no. 1, pp. 132–151, Jan. 2012.
- [5] H. Y. Zhang, C. Urtado, and S. Vauttier, “Architecture-centric component-based development needs a three-level ADL,” in *Proc. of the 4th ECSA conf.*, ser. LNCS, vol. 6285. Copenhagen, Denmark: Springer, August 2010, pp. 295–310.
- [6] H. Y. Zhang, L. Zhang, C. Urtado, S. Vauttier, and M. Huchard, “A three-level component model in component-based software development,” in *Proc. of the 11th GPCE Conf.* Dresden, Germany: ACM, Sept. 2012, pp. 70–79.
- [7] A. Mokni, M. Huchard, C. Urtado, S. Vauttier, and H. Y. Zhang, “Towards automating the coherence verification of multi-level architecture descriptions,” in *Proc. of the 9th ICSEA*, Nice, France, Oct. 2014, pp. 416–421.
- [8] —, “Formal rules for reliable component-based architecture evolution,” in *Formal Aspects of Component Software - 11th International FACS Symposium revised selected papers*, Bertinoro, Italy, Sept. 2014, pp. 127–142.
- [9] M. Leuschel and M. Butler, “ProB: An Automated Analysis Toolset for the B Method,” *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 2, pp. 185–203, Feb. 2008.
- [10] J.-R. Abrial, *The B-book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
- [11] M. Leuschel and J. Bendisposto, “Directed model checking for B: An evaluation and new techniques,” in *Formal Methods: Foundations and Applications*, ser. Lecture Notes in Computer Science, J. Davies, L. Silva, and A. Simao, Eds. Springer Berlin Heidelberg, 2011, vol. 6527, pp. 1–16.
- [12] N. Medvidovic and R. N. Taylor, “A classification and comparison framework for software architecture description languages,” *IEEE TSE*, vol. 26, no. 1, pp. 70–93, Jan. 2000.
- [13] N. Medvidovic, “ADLs and dynamic architecture changes,” in *Joint Proc. of the Second International Software Architecture Workshop and International Workshop on Multiple Perspectives in Software Development on SIGSOFT ’96 Workshops*. New York, USA: ACM, 1996, pp. 24–27.
- [14] R. Allen and D. Garlan, “A formal basis for architectural connection,” *ACM TOSEM*, vol. 6, no. 3, pp. 213–249, Jul. 1997.
- [15] J. Magee and J. Kramer, “Dynamic structure in software architectures,” *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 6, pp. 3–14, 1996.
- [16] J. Aldrich, C. Chambers, and D. Notkin, “Archjava: connecting software architecture to implementation,” in *Proc. of the 24th ICSE Conf.*, May 2002, pp. 187–197.
- [17] H. Y. Zhang, C. Urtado, S. Vauttier, L. Zhang, M. Huchard, and B. Coulette, “Dedal-CDL: Modeling first-class architectural changes in Dedal,” in *Proc. of the Joint 10th WICSA and 6th ECSA conf.*, Helsinki, Finland, August 2012.

Using Learning Styles of Software Professionals to Improve their Inspection Team Performance

Anurag Goswami¹, Gursimran Walia², Abhinav Singh³
Department of Computer Science, Office of International Services
North Dakota State University^{1, 2}, Indiana University³
anurag.goswami@ndsu.edu¹, gursimran.walia@ndsu.edu², singhab@iu.edu³

Abstract— Inspections of software artifacts during early software development aids managers to detect early faults that may be hard to find and fix later. While inspections are effective, evidence suggests that inspection abilities of individuals vary widely which affect overall inspection effectiveness. Cognitive psychologists have used Learning Styles (LS) to measure an individual's characteristic strength and ability to acquire and process information. This concept of LS is being utilized in software engineering domain as a means to improve inspection performance. This paper presents the results from an industrial empirical study, wherein the LS's of individual inspectors were manipulated to measure its impact on the fault detection effectiveness of inspection teams. Using inspection data from nineteen professional developers, we developed virtual teams with varying LS's of individual inspectors and analyzed the team performance. The results from the current study show that, teams of inspectors with diverse LS's are significantly more effective at detecting faults as compared to teams of inspectors with similar LS's. Therefore, LS's can aid software managers to create high performance inspection team(s) and manage software quality.

Keywords—software inspection; learning style; requirements.

I. INTRODUCTION

Inspecting early lifecycle artifacts (e.g., requirements and design) can improve software quality by helping developers detect faults early in the *Software Development Life Cycle* (SDLC). Empirical evidence showed that, finding and fixing faults earlier rather than later is easier, less expensive and saves significant rework costs [1]. To have most impact on software quality, researchers and practitioners have focused efforts on finding and fixing faults committed during the requirements development [2]. Requirements development is the first and a critical phase, wherein requirements are gathered from different technical (developers, designers, testers) and non-technical (managers, end-users) stakeholders. These requirements are recorded using *Natural Language* (NL) in a *Software Requirements Specification* (SRS) document. SRS is a means of communications amongst stakeholders but is prone to *mistakes* and *faults* due to inherently ambiguity, imprecision and vagueness in NL [3].

Among different approaches used for detecting NL requirement faults (e.g., NL to State transitions [4], checklist based inspections [5], scenario based reading [6], ad hoc inspections [7]), software inspections are widely recognized as most effective technique. Inspection process includes reviewing a software work-product by a group of skilled individuals to identify faults. Empirical evidence demonstrate the benefits of inspection on artifacts developed at all phases of development (e.g., requirement, design, code, interfaces) [8].

The phases in the inspection process defined by Fagan [9] are: 1) selecting skilled individuals/inspectors; 2) individual review to find faults; 3) team meeting to consolidate faults; 4) follow-up and repair. Fagan [9] emphasized different parts of the process (e.g., more emphasis on an individual preparation phase rather than team meeting phase). Regardless of the team meetings, evidence shows that the effectiveness (# of faults found) of an inspector during the individual review significantly impacts the overall effectiveness of an inspection team [10].

To improve the performance of inspectors during the individual review, researchers have tried to understand whether individual factors (e.g., educational background; level of technical degree) are correlated to their inspection effectiveness [11]. Contrary to the expectations, results at major software organizations showed that software engineers with a non-technical degree found significantly more number of requirement faults as compared to the technical degree holders [11]. Even when inspectors use same technique, and receive same training, their effectiveness varies significantly. These results led us to hypothesize that inspector's ability of detecting faults in a software artifact are affected by the ways with which they psychologically acquires, process and retains information (as opposed to their technical expertise and level of education).

On that end, cognitive psychology research [12] affirms that individuals vary in their abilities to perceiving and process information, i.e. they have varying *Learning Style* (LS) preferences and strengths. For example, some people like to think and work alone; some are more comfortable learning through concrete evidence and examples. Research results of LS in psychology prove that an individual perceive and process information better if it is presented in their preferred LS [13]. Our research extends the idea of individual LS's to evaluate its impact on the software inspection process.

While the concept of using LS in software engineering domain is novel, academia have experimented with creating heterogeneous teams to improve team performance [14]. Software Engineering researchers have also borrowed psychology research to improve inspection team performance [15]. As an example, researchers' used *Myers-Briggs Type Indicator* (MBTI) instrument (that measures psychological preference of individuals) to create heterogeneous inspection teams to maximizing disparity between team members. Despite these novel efforts, they have met with limited success because unlike LS instruments (that measure the learning preferences), MBTI is a personality inventory [16]. The only research linking LS's in software engineering domain [17] have been at studying the communication aspects of the stakeholders during requirements elicitation. Their research has shown that LS's of non-technical stakeholders should be considered when selecting

the requirements elicitation methods. The results also showed that software engineers (like other human beings) have different learning preferences. This result motivated us to evaluate if LS can aid in planning and performing the inspection.

We hypothesize that inspector’s *Learning Styles* (LS) can be used to create heterogeneous inspection teams which in turn, would increase their team performance by detecting more unique faults (i.e. less fault overlap) during the inspection. To evaluate this hypothesis, this paper presents results of an industrial study on the effect of LS preferences of nineteen professional software engineers on their inspection team performance. The participants reported their LS’s and individually inspected a requirements document using the fault-checklist technique and recorded faults. We analyzed the impact of LS’s of inspectors by creating virtual inspection teams (by combining individual data) for different team sizes. Next, all virtual teams for each team size were sorted from most dissimilar to most similar in terms of the LS’s of individual inspectors followed by an evaluation of their team performances. The results show that team of inspectors with dissimilar LS’s performed significantly better than the teams of inspectors with similar LS’s. Software managers can use these results to plan and manage inspections in their organizations.

II. BACKGROUND - MEASURING LEARNING STYLES

Kolb [18] introduced the concept of LS’s, and developed the first LS instrument. Over the years, psychologists have developed different versions of LS models [19] and validated the use of LS’s in engineering education [12]. Previous researches revealed that the *Felder and Silverman’s Learning Style Model* (FSLSM) is the most advanced and widely used to measure the LS’s preference of individuals [20]. The instrument used to measure LS is known as the *Index of Learning Styles* (ILS) [12] and is used in this research as below.

A. Felder and Silverman Learning Style Model (FSLSM)

The FSLSM model (shown in Fig. 1) capture most important LS preferences among individuals and then classifies characteristic strength and preference across four LS dimensions. These dimensions related to the way individuals “perceive” and “process” information. The two dimensions which relates to perceiving information includes: a) *Sensing/Intuitive*; and b) *Visual/Verbal*. The remaining two dimensions (i.e. *Active/Reflective* and *Sequential/Global*) relate to information processing. Brief description of LS model is described in Fig. 1.

We have used FSLSM and its accompanying instrument, *Index of Learning Style* (ILS), to measure the LS of inspectors.

B. Index of Learning Styles (ILS)

The ILS instrument has been empirically validated for its reliability and construct validity [21]. A sample ILS output is shown in Fig. 2. The ILS instrument is an online questionnaire with 44 questions. Each LS dimension has 11 questions. For example, in *Visual/Verbal* dimension, if a person selects 10 answers that favors visual category and 1 towards verbal category then the LS score will be 9 (i.e. 10-1) with a ‘strong’ preference towards the visual category represented by a symbol ‘X’ on the top of the score (see Fig. 2). The symbol ‘X’ represents the preference towards a category in a LS dimension.

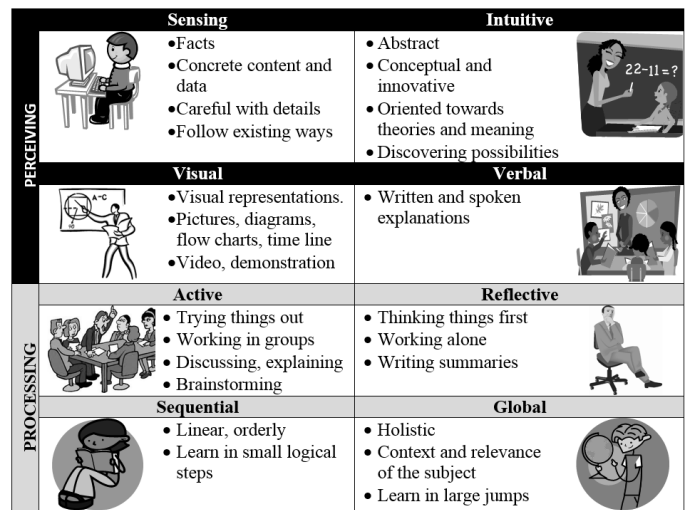


Figure 1. Felder Silverman Learning Style Model

ILS score ranging from 1 to 3 represents that a person is balanced towards both the categories in an LS dimension. A score between 5-7 and 9-11 states that the person has a *moderate* and *strong* preference towards a category in a LS dimension.

III. RESEARCH APPROACH – USING LS TO FORM VIRTUAL INSPECTION TEAMS

The goal of the study was to be evaluate the impact of LS’s on the inspection performance by creating multiple virtual inspection teams for varying number of inspectors (e.g., ranging from N=2 to N=10 inspectors) and then sorted from most dissimilar to most similar w.r.t the LS’s of team members. This was followed by an evaluation of their inspection performances (*effectiveness*). To achieve this objective, a software tool was developed to automate this process by utilizing different multivariate statistical approaches. These approaches are briefly described here with more details in [22].

A. Principal Component Analysis (PCA)

FSLSM classifies each LS dimension into two categories (sensing/intuitive, visual/verbal, active/reflective and sequential/global). The relationship between two categories of each dimension is negatively correlated. That is, as score in one category increases, score in the other decreases. The **first** step was to transform the original correlated variables (i.e. LS scores across categories in each LS dimension) into uncorrelated variables. PCA is a multivariate technique that is being used to convert a set of observations of possibly correlated variables into set of values of uncorrelated variables called principal

ACT	11	9	7	5	3	1	1	3	5	X	7	9	11	REF
						<--	-->							
SEN	11	9	7	5	3	1	1	3	5	7	9	11	INT	
						<--	-->							
VIS	11	X	9	7	5	3	1	1	3	5	7	9	11	VRB
						<--	-->							
SEQ	11	9	7	5	3	X	1	1	3	5	7	9	11	GLO
						<--	-->							

Figure 2. Example result of the questionnaire on the ILS

components (PCs) [23]. PCA is used in this research to better understand the interrelationships between two categories (e.g., visual/verbal) of each of the four LS dimensions and between all the four LS dimensions for each individual. PCA transforms the original correlated data (i.e., FSLSM output, Fig. 2) into a new set of uncorrelated variables called principal components (PCs) [23]. **Note**, for each individual in our research, the numbers of possible PC's are always equal to or less than the number of original variables (i.e., 8 categories across 4 LS dimensions)[24].

B. Cluster Analysis (CA)

During the **second** step, CA was used to group similar participants into different clusters based on their LS's. CA [23] is being used in our research to form groups/clusters of individuals based on their LS data. The resulting clusters of CA explain high similarity of LS's within each cluster and high dissimilarity of LS's between different clusters [25]. Among different types of clustering techniques (e.g. Hierarchical, Non-Hierarchical, Agglomerative, Divisive clustering); we have used k-means clustering algorithm [26]. CA groups the participants into clusters of similar LS, which helped us to study the relation between LS of members on the scale ranging from dissimilar to similar LS preferences. A team formed with different cluster members will lead to dissimilar LS group and a team formed from same cluster members leads to a similar LS group. More details of CA can be found here [22].

C. Discriminant Analysis (DA)

During the **third** step, DA was used to find out the probability of a participant belonging to a cluster. Using DA, LS variations are partitioned into a "between group" and a "within-group". This result of the DA is used to maximize the LS variations across different clusters, and minimize the LS variations within each cluster [27]. While CA explained that there is more dissimilarity among different clusters, there is a lack of dissimilarity in the LS preferences of the individuals belonging to the same cluster. DA provides *Group Membership* (GM) to determine the dissimilarities between individual LS's within the same cluster and with respect to the individuals in other clusters. So, DA provides GM values for each individual w.r.t each cluster. GM was used in our study to sort the teams ranging from most dissimilar LS to most similar LS preferences and strengths.

This process of extracting software inspection teams with varying levels of LS preferences was automated. Details of evaluating the performance of these teams appears in Section V.

IV. EMPIRICAL STUDY DESIGN

To evaluate the impact of LS variability on the inspection performance, LS's of nineteen professionals (working in IT Company) were gathered via online survey questionnaire. The participating subjects were trained on the inspection process and on using the fault checklist to record faults found during the inspection. Next, each subject individually inspected an industrial strength requirements document (that was seeded with faults) and reported faults. Study details are provided below.

Research Question: Whether the variation in the LS's of individual inspectors is positively correlated to their team performance during an inspection of requirements document?

Variables: The study manipulated the LS's of individual inspectors (*independent variable*) and measured its effect on the team effectiveness (*dependent variable*) during the inspection.

Participating Subjects: Nineteen software professionals working in a software company participated in the study. Some of them have worked on multiple projects in industry. The subjects' reported to have an average of three years of experience in interacting with user to writing and inspecting requirements and use cases.

Artifacts: The document inspected in the study described the requirements for the Loan Arranger system (LAS). LAS is responsible for grouping loans into bundles based on user-specified characteristics and then sell to other financial institutions. For use in previous studies, the document was written in plain English, was 10 page long, and seeded with thirty realistic faults. The fault seeding was done by Microsoft researchers prior to the study. The document is publicly available¹ and have been used in several inspection studies [28, 29].

Experiment Procedure: Study steps as described below:

Step 1 – Pre study survey: participants were asked to fill pre-study survey questionnaire to provide feedback about their experience of working in software industry. The survey elicited information about their experience in interacting with end-users to write requirements, writing use cases, inspecting requirements, and changing requirements for maintenance.

Step 2 – Learning Style Questionnaire Survey: participants were given Felder Silverman's LS questionnaire. Participants answered 44 multiple choice questions² and, the LS results are generated for each participant on ILS scale (Fig. 2). For each dimension on ILS (Active/Reflective, Sensing/Intuitive, Visual/Verbal, and Sequential/Global), the participant has score towards one category. Hence, only four LS categories (from each dimension) form LS of an individual with a score of either 1 or 3 or 5 or 7 or 9 or 11. These scores are then converted into actual scores which has scores in both the categories (i.e. number of answers supported for each category in a dimension) as shown for a subset of 10 (out of 19) subjects in Table I. For example, in Active/Reflective dimension, subject ID 9 answered 8 questions in favor of Active and 3 in favor of Reflective.

TABLE I. EXAMPLE OF ACTUAL SCORES OF 10 PARTICIPANTS

ID	ACT	REF	SEN	INT	VIS	VER	SEQ	GLO
1	5	6	8	3	9	2	6	5
2	5	6	7	4	6	5	6	5
3	9	2	9	2	10	1	5	6
4	3	8	7	4	4	7	7	4
5	4	7	10	1	11	0	6	5
6	4	7	5	6	9	2	4	7
7	5	6	8	3	11	0	9	2
8	3	8	5	6	4	7	10	1
9	8	3	6	5	8	3	3	8
10	4	7	10	1	6	5	6	5

¹ <http://steel.cs.ua.edu/~carver/BackgroundReplication>

² <https://www.engr.ncsu.edu/learningstyles/ilsweb.html>

Step 3 – Training and Inspecting LAS Requirements: The subjects were trained (by the same researcher in a single session) on basic concepts in an inspection and how to detect faults in a requirements document using fault checklist technique. The subjects were instructed on different fault types and how to use the fault form to record faults during the inspection using example requirements. Then, the subjects were asked to work alone and performed an inspection of LAS to identify and record faults. To normalize the results, the subjects were provided sixty minutes to perform the inspection (read the document and record faults). At the end of inspection process, nineteen fault lists were collected (one per subject). One of the researchers read through the faults reported by each participant (and compared against the seeded fault list) to remove any false-positives before analyzing the data. In addition, the fault reporting forms required the subjects to classify the faults identified during the inspection into one of the following fault types: *Omission* (O), *Ambiguous Information* (A), *Inconsistent Information* (II), *Incorrect Fact* (IF), *Extraneous* (E), and *Miscellaneous* (M).

V. EVALUATION CRITERIA

This section describes the process used to form virtual inspection teams (using individual LS data), sorting teams (w.r.t LS dissimilarity) and evaluating their team performance (using individual fault data). Our previous results showed that, for cost-effective inspections, team sizes should be limited to ten inspectors due to high cost of inspections and diminishing return beyond a size of 10 inspectors [30]. Therefore; the tool generated all possible virtual inspection teams for inspection team size from N=2 to N=10 inspectors. For each inspection team size, the tool sorts the virtual teams in the decreasing order of LS dissimilarity of the inspectors. The tool then outputs the total unique faults found by each team and their fault detection rate. The evaluation steps are described in subsections V.1-4.

1) *Creating Virtual inspections:* We created virtual inspection teams (i.e. teams that did not actually meet, we just combined their data) with team size ranging from 2 to 10 inspectors and each team size has all the possible combinations of virtual teams. For example, to create virtual inspection team of size 4 (from a pool of 19 inspectors), we created 3876 virtual inspection teams (i.e. $19C_4$).

2) *Grouping of similar inspectors in clusters:* The correlated LS of inspectors in each LS dimension were converted into uncorrelated variables by the tool using PCA (Section III.A). Next, inspectors of similar LS's were grouped together in the same cluster (number of cluster is same as the team size being analyzed) using CA (Section III.B).

3) *Sorting teams based on the LS of inspectors:* In this step, the tool calculates the group membership (GM) of each inspector in a cluster using DA. Next, tool sorts all inspection teams (i.e. $32C_4$, from step 1) in the order of decreasing level of dissimilarity (i.e. most dissimilar to similar) in the LS's of the individual inspectors.

4) *Evaluating inspection performance of teams:* During this step, the tool combines the individual inspection data and outputs the total unique faults and average time taken by each

virtual inspection team of all sizes. To summarize, all possible inspection team were formed for each team size and their virtual inspection results were organized from teams with dissimilar to similar LS's along with their performance.

VI. DATA ANALYSIS AND RESULTS

This section presents the results on the; 1) effect of variation in the LS's on the inspection team performance; 2) distribution of fault types (mentioned in Section IV) across different LS dimension and categories.

As stated earlier, for each team size (e.g., N=4), all possible virtual teams were generated and then sorted with dissimilar LS's (highest number of cluster involved in team formation) to similar (least number of cluster involved) LS's. Inspection data (i.e. faults found by each participant) was individual data; so fault detection *effectiveness* for virtual teams was calculated by combining the unique faults detected by each participant in LAS requirements document. This analysis was performed on all possible virtual inspection teams for all sizes.

Fig. 3 compares the average number of unique faults found by virtual inspection teams ranging from N= 2 to 10 inspectors. Each line represents a particular team size (e.g., N=4) and maps the average number of faults found by the virtual inspection teams (formed with a certain # of clusters). Results are organized by the increasing number of clusters (or increasing dissimilarity) involved in the team formation (i.e., the higher the cluster number, the more dissimilarity the team members). Also mentioned earlier, the # of clusters that could participate in the team formation is always less than or equal to the team size (e.g., 1 or 2 or 3 clusters for team size 3).

Based on the results in Fig. 3, a general observation is that, for each team size, teams with highest number of clusters involved (i.e. most dissimilar inspectors) found maximum average number of faults as compared to the same team size with less number of clusters. The results show a consistent increase in the inspection effectiveness with an increase in the number of clusters used to form teams. For example, in team size 5, teams created with only one cluster (i.e. most similar teams) found an average of 13.42 faults; whereas teams created from five different clusters (i.e. most dissimilar team) found an average of 17.45 faults. This effectiveness trend is consistent across all team sizes.

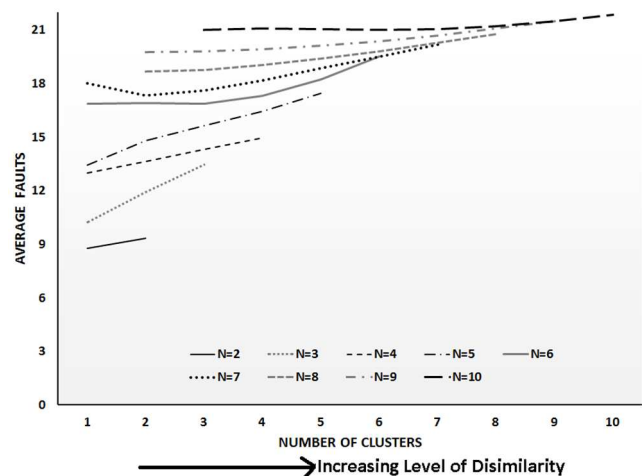


Figure 3. Team effectiveness organized by increasing # of clusters

As an exception (in Fig. 3), for some larger team size (e.g., team size 10), virtual inspection teams could not be formed from a single cluster. This is because for team size 10, the tool creates 10 different clusters via k-means algorithm. All 19 participants were distributed across these 10 clusters and there was no cluster that contained all 10 participants. Therefore, as the team size increases, the number of participants that belong to the same cluster decreases which reduces the probability that a team will be formed from less number of clusters

Based on the above results, teams of inspectors with dissimilar LS's had less fault overlap and consequently, their inspection effectiveness is higher.

To evaluate this effect, we performed a linear regression test to see whether the dissimilarity in the LS's of inspectors is positively correlated with the average number of unique faults found by inspection teams of different sizes. The results (Table II) show that, dissimilarity in the LS of inspectors had a strong and significant positive correlation with the team effectiveness for team size 3 to 10 (shaded rows). We anticipate that increasing the team size beyond a certain number of inspectors would not significantly diversify the LS's of inspectors in a team (due to 8 possible LS categories). Generally, software companies do not

TABLE II. LINEAR REGRESSION RESULTS

Team size	Effectiveness Correlation
2	$P=0.14$; $Correl. Coeff = 0.112$; $r^2=0.013$
3	$P<0.001$; $Correl. Coeff = 0.387$; $r^2=0.015$
4	$P<0.001$; $Correl. Coeff = 0.185$; $r^2=0.034$
5	$P<0.001$; $Correl. Coeff = 0.268$; $r^2=0.072$
6	$P<0.001$; $Correl. Coeff = 0.228$; $r^2=0.052$
7	$P<0.001$; $Correl. Coeff = 0.292$; $r^2=0.085$
8	$P<0.001$; $Correl. Coeff = 0.211$; $r^2=0.044$
9	$P<0.001$; $Correl. Coeff = 0.166$; $r^2=0.028$
10	$P<0.001$; $Correl. Coeff = 0.060$; $r^2=0.004$

employ a large inspection teams (which is the reason we had analyzed up to team of 10 inspectors). Overall, creating inspection teams based on the dissimilarity in their LS strengths (guided by the number of clusters involved in their formation) appears to increase the fault detection effectiveness during an inspection of requirements document.

We analyzed the impact LS dimensions (made up of combination of categories) had, on inspection effectiveness and nature of fault found. The was done to gain insights about whether certain LS are responsible for higher inspection effectiveness and detection of particular fault type or distributed across different LS dimension? To perform this analysis, we captured the classification of faults according to their fault type (Section IV) found by the participants belonging to each cluster. From raw data it was found, none of the inspectors had a preference towards *Verbal-VER*. The remaining LS categories (*Active-ACT*, *Reflective-REF*, *Sensing-SEN*, *Intuitive-INT*, *Sequential-SEQ*, and *Global-GLO*) were analyzed.

To analyze the effect of each LS (i.e. combination of categories across each dimension) on inspection effectiveness and fault types, we created groups of each LS using six LS categories across three LS dimensions. Therefore, eight clusters with their respective number of members were: *REF-SEN-GLO* (five), *ACT-SEN-SEQ* (five), *ACT-INT-SEQ* (one), *REF-SEN-SEQ* (one), *REF-INT-SEQ* (two), *ACT-SEN-GLO* (one), *ACT-INT-GLO* (two), and *REF-INT-GLO* (one). Fig. 4 shows the

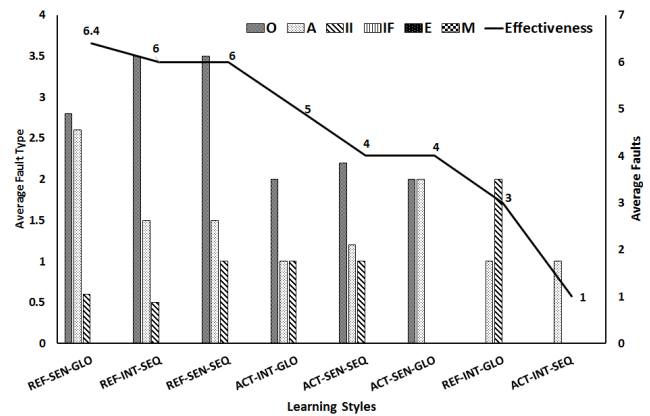


Figure 4. Effectiveness and Fault Type by each LS

average inspection effectiveness (shown by solid line) and average fault type (shown by bars) correlation for each LS cluster during inspection. Results are ordered from most effective to least effective cluster. Left y-axis shows the average number of different fault type detected and secondary y-axis on the right shows the average inspection effectiveness.

Based on results in Fig 4, following observations were made:

- Inspectors with REF-SEN-GLO LS's had the maximum inspection effectiveness (6.4) and close to that, REF-INT-SEQ LS cluster found the next maximum effectiveness (6). Upon analyzing the pre-study survey data, it was found that inspectors with REF-SEN-GLO LS preference had high experience of working with requirements analysis as compared to inspectors with REF-INT-SEQ LS preference.
- Inspectors belonging to the REF-SEN-GLO (i.e. most effective cluster) found the maximum number of Ambiguous Information fault (A). This result suggests that inspection performance rely on a combination of categories along each LS dimension and a single LS category cannot help detect all types of faults. Hence, there should be an inspection team with inspectors of diverse LS's.
- Another observation is that REF-INT-SEQ and REF-SEN-SEQ cluster found the maximum number of Omission (O) faults. Also, REF-INT-GLO cluster found maximum number of Inconsistent Information (II) faults.

These results reinforce that, **a combination of different LS's enabled inspectors to find faults of different types** and that **difference in the LS of inspectors enable a higher coverage of faults present in an artifact**. The result however revealed that inspectors belonging to the five LS clusters (i.e., *REF-SEN-GLO*, *REF-INT-SEQ*, *REF-SEN-SEQ*, *ACT-INT-GLO*, *ACT-SEN-SEQ*) out of 8 clusters were able to uncover all the three types (O, A, and II) of faults present in requirements document.

VII. THREATS TO VALIDITY

In this experiment, we were able to address some of the validity threats. Participating subjects were software professionals working in real industry settings. Heterogeneity of document was handled by providing participants with the same LAS document to inspect. Our experiment consists of inspectors who has different levels of work experience due to which group composition effect was not addressed. Training was provided by one trainer to all the participating subjects which addressed the training bias. We were also able to address

fatigue effect by providing enough time to participants to take surveys (i.e. LS questionnaire, pre-study, and post-study survey) and perform requirements inspection in their comfortable environment where they can take break(s). However, the LAS document was developed externally by Microsoft and we do not have access of LS of authors which did not led us to control the effect of LS of authors of SRS on inspection output. Also, for larger team size (e.g., team size 10), there was not enough data to form virtual inspection teams with small number of clusters. These issues regarding the generalization of results will be addressed in future studies.

VIII. DISCUSSION OF RESULTS

The focus of our study was to investigate the impact of individual inspector on performance of inspection team during the requirements inspection. The results from Section VI (Fig. 3) showed that dissimilarity in LS of inspectors had a direct and positive relationship with inspection team effectiveness. This means, higher the dissimilarity in the LS of inspectors in a team, the more number of faults are detected in requirements document during the inspection (i.e. higher inspection output). While testing this results statistically (Table II), there was a strong significant correlation between the LS dissimilarity and inspection effectiveness. Therefore, using LS's as an input to guide staffing/formation of inspection teams is beneficial for software managers. Results also revealed that some LS do favor inspection positively (i.e. high effectiveness and on detection of different fault type) as compared to other LS's. Based on the results provided in this paper, the concept of LS is applicable in software inspections domain and can help to manage the quality of software by creating high performance inspection team(s).

IX. CONCLUSION AND FUTURE WORK

While the data size used in this study was small, these results showed that if inspection teams are created by taking different LS of inspectors into account, they would read requirements document with different perspectives. This results in less fault overlap among inspectors in a team and leads to high inspection output. These results are interesting and provide us with initial evidence to continue on this path. We plan to analyze the pre/post study data to gain more insights into the LS's of professional developers and its impact on their daily activities. Our future works includes replicating this analysis for larger data sets. Another future work includes analyzing the data to evaluate the correlation (positive or negative) inspectors with certain LS preferences (e.g., *Active vs. Reflective*) may have on their performance during the requirements inspections.

REFERENCES

- [1] Perry, W.E.: 'Effective Methods for Software Testing: Includes Complete Guidelines, Checklists, and Templates' (John Wiley & Sons, 2006. 2006)
- [2] Ackerman, A.F., Buchwald, L.S., and Lewski, F.H.: 'Software inspections: an effective verification process', *Software, IEEE*, 1989, 6, (3), pp. 31-36
- [3] Berry, D.M., and Kamsties, E.: 'Ambiguity in requirements specification': 'Perspectives on software requirements' (Springer, 2004), pp. 7-44
- [4] Aceituna, D., Do, H., Walia, G.S., and Lee, S.-W.: 'Evaluating the use of model-based requirements verification method: A feasibility study'. *Empirical Requirements Engineering (EmpiRE)*, 2011 First International Workshop on 2011 pp. 13-20
- [5] Parnas, D.L., and Lawford, M.: 'The role of inspection in software quality assurance', *Software Engineering, IEEE Transactions on*, 2003, 29, (8), pp. 674-676
- [6] Shull, F., Rus, I., and Basili, V.: 'How perspective-based reading can improve requirements inspections', *Computer*, 2000, 33, (7), pp. 73-79
- [7] Porter, A.A., Votta Jr, L.G., and Basili, V.R.: 'Comparing detection methods for software requirements inspections: A replicated experiment', *Software Engineering, IEEE Transactions on*, 1995, 21, (6), pp. 563-575
- [8] Fagan, M.E.: 'Design and code inspections to reduce errors in program development': 'Pioneers and Their Contributions to Software Engineering' (Springer, 2001), pp. 301-334
- [9] Fagan, M.E.: 'Advances in software inspections': 'Pioneers and Their Contributions to Software Engineering' (Springer, 2001), pp. 335-360
- [10] Porter, A., Siy, H., Mockus, A., and Votta, L.: 'Understanding the sources of variation in software inspections', *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1998, 7, (1), pp. 41-79
- [11] Carver, J.: 'The impact of background and experience on software inspections', *Empirical Software Engineering*, 2004, 9, (3), pp. 259-262
- [12] Felder, R.M., and Silverman, L.K.: 'Learning and teaching styles in engineering education', *Engineering education*, 1988, 78, (7), pp. 674-681
- [13] Allert, J.: 'Learning style and factors contributing to success in an introductory computer science course', (*IEEE*, 2004), pp. 385-389
- [14] Rutherford, R.H.: 'Using personality inventories to help form teams for software engineering class projects', *SIGCSE Bull.*, 2001, 33, (3), pp. 73-76
- [15] Miller, J., and Yin, Z.: 'A cognitive-based mechanism for constructing software inspection teams', *Software Engineering, IEEE Transactions on*, 2004, 30, (11), pp. 811-825
- [16] Montgomery, S.M.: 'Addressing diverse learning styles through the use of multimedia'. *Frontiers in Education Conference*, 1995. Proceedings., 1995 pp. 3a2. 13-13a12. 21 vol. 11
- [17] Aranda, G.N., Vizcaino, A., Cechich, A., and Piattini, M.: 'A cognitive-based approach to improve distributed requirements elicitation processes'. *Cognitive Informatics, 2005. (ICCI 2005). Fourth IEEE Conference on*, Irvine, USA, 8-10 Aug. 2005 pp. 322-330
- [18] Kolb, D.A.: 'Experiential learning: Experience as the source of learning and development' (Prentice-Hall Englewood Cliffs, NJ, 1984. 1984)
- [19] Charkins, R., O'Toole, D.M., and Wetzel, J.N.: 'Linking teacher and student learning styles with student achievement and attitudes', *Journal of Economic Education*, 1985, pp. 111-120
- [20] Felder, R.M.: 'Are learning styles invalid?(Hint: No!)', *On-Course Newsletter*, 2010, pp. 1-7
- [21] Felder, R.M., and Spurlin, J.: 'Applications, reliability and validity of the index of learning styles', *International Journal of Engineering Education*, 2005, 21, (1), pp. 103-112
- [22] Goswami, A., and Walia, G.: 'Using Learning Styles to Create Virtual Inspection Teams: A Technical Report', <http://www.goswamianurag.com/techRep/LSTeamsTech.pdf>, The Department of Computer Science, North Dakota State University, 2015
- [23] Anderson, T.W.: 'An introduction to multivariate statistical analysis' (Wiley New York, 1958.)
- [24] Jolliffe, I.T.: 'Principal component analysis' (Springer verlag, 2002. 2002)
- [25] Steinbach, M., Ertöz, L., and Kumar, V.: 'The challenges of clustering high dimensional data': 'New Directions in Statistical Physics' (Springer, 2004), pp. 273-309
- [26] Hartigan, J.A., and Wong, M.A.: 'Algorithm AS 136: A k-means clustering algorithm', *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1979, 28, (1), pp. 100-108
- [27] Tatsuoaka, M.M., and Tiedeman, D.V.: 'Chapter IV: Discriminant Analysis', *Review of Educational Research*, 1954, 24, (5), pp. 402-420
- [28] Carver, J., Shull, F., and Basili, V.: 'Observational studies to accelerate process experience in classroom studies: an evaluation'. *Empirical Software Engineering*, 2003. ISESE 2003. Proceedings. 2003 International Symposium on 2003 pp. 72-79
- [29] Shull, F., Carver, J., and Travassos, G.H.: 'An empirical methodology for introducing software processes', *ACM SIGSOFT Software Engineering Notes*, 2001, 26, (5), pp. 288-296
- [30] Mandala, N.R., Walia, G.S., Carver, J.C., and Nagappan, N.: 'Application of kusumoto cost-metric to evaluate the cost effectiveness of software inspections'. *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, Lund, Sweden, 17-22 Sep. 2012 pp. 221-230

How do software engineers apply an early usability inspection technique? A qualitative study

Natasha Malveira C. Valentim, Tayana Conte
USES Research Group, Universidade Federal do Amazonas
Manaus, Brazil
{natashavalentim, tayana}@icomp.ufam.edu.br

Bernardo Estácio, Rafael Prikkladnicki
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Brazil
{bernardo.estacio, rafaelp}@puccrs.br

Abstract — Usability inspections can be employed in early phases of the software development process. They improve usability through artifacts that are built during the development of the software. These artifacts will influence the usability of the developed software. Usability inspection techniques have been proposed and considered as an effective alternative for addressing usability issues in early phases. However, these techniques are often avoided by software engineers due to their lack of experience and knowledge in the field. Therefore, there is an opportunity to investigate how industry practitioners have employed an early usability inspection technique in practice. This paper describes an observational study in the industry aimed at eliciting the process used by software engineers when applying an early usability inspection technique. We analyzed the qualitative data, discussing their impact in the improvement of the technique. The results indicated which steps the software engineers adopted in the technique's application.

Keywords- Usability evaluation, usability inspection; early usability; qualitative study.

I. INTRODUCTION

Usability is universally acknowledged as a significant aspect of the overall quality of interactive systems [1]. Including usability allows benefits such as improving user productivity and reducing training and documentation costs [2]. Therefore, a large number of researchers have investigated ways to include usability in software development [3], [4]. Donahue [5] says that investments in usability have allowed benefits such as income increase. This has motivated more organizations to consider usability as a relevant factor in their software products [6].

However, Seffah and Metzker [7] highlight some of the following challenges when including usability into the development process: (a) usability activities are usually separated from the software development process, and (b) the notations and tools in which usability is considered are different from those employed in the development process. Furthermore, the development processes do not take advantage of the intermediate artifacts that are produced during early stages (i.e., requirements and design stages). These intermediate artifacts (e.g., navigational models) are mainly employed to guide software engineers and to document the application. Since the traceability between these artifacts and the final application is not well defined, performing evaluations using these artifacts can be difficult [8].

For this reason, it is important to propose technologies that can be applied by software engineers in the usability evaluation of the artifacts that are employed in the early stages of the development process. The benefits of using this type of technologies are: (i) to assist developers in learning about usability and interaction design and (ii) to reduce the usability evaluation costs because often these evaluations are performed only when the software is ready, generating rework and increasing costs with repairs and improvements.

In this context, this paper describes an observational study with 15 software engineers who applied a usability inspection technique in mockups. They had between 1 and 13 years of experience in the development of projects with the software industry, both Web applications and Mobile applications. In this observational study, we intended to 'look inside' the inspection process, enabling us to understand how software engineers apply an early usability inspection technique.

The early usability inspection technique analyzed in this study is called MIT 2. This technique aims at evaluating the usability through mockups [9]. The MIT 2 is part of a set of techniques called Model Inspection Techniques for Usability Evaluation (MITs), composed by two other techniques: MIT 1 (for the usability evaluation of use case specifications) and MIT 3 (for the usability evaluation of activity diagrams). The MITs intend to reduce the cost of fixing usability problems in artifacts that are employed in the early stages of the development process. They have verification items that guide the software engineers in the discovery of usability problems. Therefore, investigating how software engineers apply this technique during an early usability inspection is important to understanding such practice.

The remainder of this paper is organized as follows: Section 2 discusses the concept of Early Usability. Section 3 presents the MITs technique. Section 4 describes the planning and the execution of the Observational Study. Sections 5 and 6 present the results of the Quantitative and Qualitative Analysis, respectively. Section 7 presents some discussions. Section 8 describes the threats to validity. Section 9 concludes the paper.

II. EARLY USABILITY

Early Usability considers the usability in early phases of the development lifecycle. The goal of Early Usability is to improve usability through artifacts that are built during the software development that will influence the quality of the developed software. Early Usability can help reduce the

number of problems detected in software development projects. It also provides benefits such as increasing the quality of the develop software and higher user satisfaction [2]. According to Fernandez et al. [8], if usability problems are repaired earlier, the quality of the final application can be improved, saving resources in the development stage. Therefore, contributing to reducing the cost of the development process.

Propp et al. [10] proposed a representative example of a technology that considers Early Usability. This approach focuses specifically in the development of interactive systems based on task models. To evaluate usability, first we use the task model to control the user interaction at a degree of abstraction based on tasks. After having introduced the usage of a task engine for task model based on capture, it is necessary to perform the connection between the initial task model and the further refined software artifacts at different stages of the development. To use this approach, it is necessary to adopt the development process based on the proposed model.

Hornbæk et al. [3] propose the UCE method (Use Case Evaluation) for the usability evaluation based on use cases that employs Nielsen's [11] heuristics as a basis. This method consists of three activities: (1) Inspection of Use Cases, that seeks to identify usability problems that the evaluator is convinced one or more prospective users will experience, (2) Assessment of Use Cases, that seeks to assess the quality of the use cases, and (3) Documentation of Evaluation, where the results are compiled into a coherent evaluation product. This method does not require computer support itself.

III. MODEL INSPECTION TECHNIQUE FOR USABILITY EVALUATION

The MITs are reading techniques to include usability in the early stages of the development process (analysis and design phase), in order for the final applications to become easier to use. According to Travassos et al. [12], reading techniques are a type of inspection technique that contains a series of steps for the individual analysis of a software product in order to achieve the necessary understanding for a specific task. Thus, the MITs main innovation is the verification items that serve as a guide to interpret Nielsen's [11] heuristics. That is, the MITs guide software engineers during the usability evaluation of Use Case specifications (MIT 1), Mockups (MIT 2) and Activity Diagrams (MIT 3). This allows software engineers to be assisted by the technique during the search for usability problems, even if they have little experience in usability.

One artifact that is often available in early stages of the software development is the mockup. It is an important artifact for both software development and for the design of user interfaces. According to Luna et al. [13], mockups are artifacts employed to represent aspects of the user interface serving as sketches of the applications. They are intended to be developed quickly to reflect the needs of customers in terms of presentation more significantly than the requirements expressed in written language. Therefore, evaluating the usability of these artifacts allows the discovery of problems early. The MIT 2 technique was built for this purpose: to assist in the discovery of problems in the initial stages of the development process, through mockups, even the early prototypes made of low-fidelity materials. It has verification items that are based on the

heuristics by Nielsen [11], but in a more guided way. That is, the verification items guide the software engineers, even if not usability experts, in the search of usability problems. The current version of MIT 2 is available in a technical report [9]. Table I presents some verification items of MIT 2.

TABLE I. PART OF THE MIT 2 TECHNIQUE [9].

MIT-2AE. Error prevention Heuristic	
Verification Item 2AE3	Verify if there is any system warning that alerts, through messages or informational texts, that what the user is doing may be inappropriate at that time;
Verification Item 2AE4	Verify if all available options, buttons and links have names that clearly define what results or conditions will be met.

The steps for using the MIT 2 technique are shown in Figure 1. These steps are: (1) to evaluate the mockup using the MIT 2 technique and (2) to identify usability problems. In order to illustrate the MIT 2's steps, we have employed it to evaluate the usability of a mockup. This mockup was created based on a page of the SION System¹. This page is used in the SION System to register a course of a training center. In the next paragraphs we describe how we applied the steps to perform a simple inspection of the mockup from the SION System. This example shows only part of the inspection of the SION System, since we are only evaluating one of its mockups.

The first step for the identification of usability problems is to evaluate the usability through verification items. In other words, software engineers must check if the mockup meets all the usability verification items described within each heuristic. Table I shows an example of the usability verification items.

In order to identify usability problems (2nd step), software engineers must point in the mockup which part did not meet the usability verification items. If we look at Figure 1 and Table I, we can relate the nonconformities of the usability verification items in Table I with the augmented element A in Figure 1.

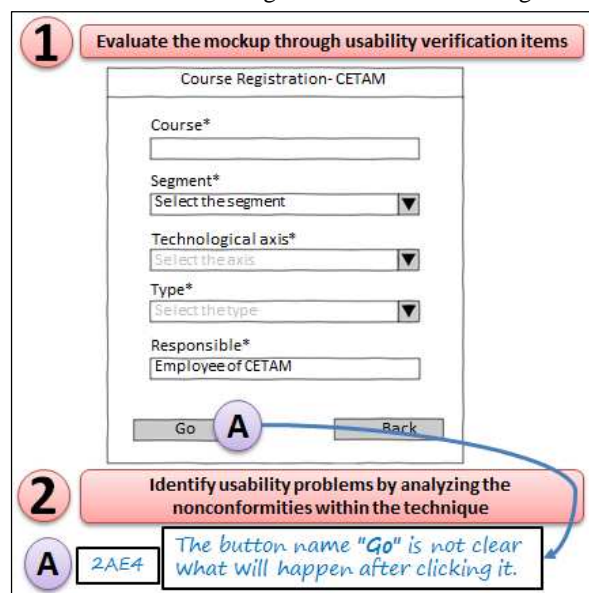


Figure 1. Example of the MIT 2's steps.

¹ <http://sion.secti.am.gov.br/principal/>.

The 2AE4 verification item requires software engineers to verify if all available options, buttons and links have names that clearly define the results that will be achieved. The name of the button “Go” does not make it clear if the course will be registered or if the user will go forward into the next screen (see Figure 1 element A). In other words, the button name does not make clear what will happen after clicking it.

IV. THE OBSERVATIONAL STUDY

To support the development and validation of the MIT 2, we have adopted the empirical methodology presented in Shull et al. [14]. It comprises four stages: (1) feasibility studies: to determine the usage possibility of the technology; (2) observational studies: to improve the understanding and the cost-effectiveness of the technology; (3) case studies in real lifecycles: to characterize the application of the technology during a real lifecycle; (4) case studies in industry: to identify if the application of the technology fits into industrial settings.

In order to verify the possibility of employing the MIT 2 technique, the authors conducted two feasibility studies. Valentim et al. [15] and Valentim and Conte [16] describe the results of the studies. The statistical test results showed that MIT 2 obtained similar effectiveness and efficiency as the Heuristic Evaluation in both the first and the second feasibility study. This indicates that further studies need to be performed to identify which part of the inspection process with the MIT 2 needs improvement. We expect that software engineers can use MIT 2 to ensure the quality of their mockups. To achieve this goal, we carried out the second stage of the methodology by obtaining a detailed understanding of how the MIT 2 is applied.

The goal of an observational study is to collect data about how a particular task is accomplished. We performed an observational study with the purpose of eliciting the process employed by software engineers when applying the current version of the MIT 2 technique. Our goal was to deeply understand the MIT 2 process, so we did not compare the MIT 2 with any other technique. The observational study should answer the following question: “Which steps the software engineers adopted in the MIT 2’s application?”.

Observational techniques can be employed to understand current work practices [17]. In this study, we gathered two types of data: observational and inquisitive data. The observational data were collected during the inspection process. To gather the observational data, we used the “Cooperative Evaluation” method [18]. In this method, the software engineer describes what (s)he is doing and the observer is free to ask questions/explanations about the software engineer’s decisions or actions [2]. Inquisitive data were gathered after finishing the inspection using interviews.

A. Planning

Quantitative data were measured in order to compare the quantitative results of this study with other studies conducted with the MIT 2 technique. The quantitative investigation points were the efficiency and effectiveness indicators of the technique. Efficiency and effectiveness were calculated for each subject as: (a) the ratio between the number of defects found and the time spent to find them; and (b) the ratio between

the number of detected defects and the total number of existing (known) defects, respectively.

The mockups used in this study are part of the SION System. The SION is a website that provides information about the advertisement and support of activities regarding Science, Technology and Innovation. The mockups that were evaluated in this study are: mockup of course registration (see Figure 1), mockup of course listing (where one can select a course to delete it or edit it), and some messages that the system displays after saving data. The mockups had real usability problems that would influence the use of the designed system.

The interviews and observations took place in a Formation Center from a large IT Company, where the focus is in innovation and software development. The center supports several software engineers in real projects, mainly in the development of Web systems and Mobile applications, adopting agile methodologies such as Scrum and XP. In order to meet ethical needs, we created a free consent form to inform about research procedures and confidentiality. Fifteen software engineers signed the consent form. All participants received one-hour training on mockups and usability principals. Examples were shown on how to use the MIT 2 technique.

The qualitative investigation points were Application Process and Intention to Use of Technique. Theses investigation points were collected during the study and were analyzed together with the data obtained from the interviews. For the interviews, a semi-structured questionnaire was used with open questions (see Table II)

TABLE II. INVESTIGATION POINTS RELATED TO THE QUESTIONS THAT WERE USED ON THE QUESTIONNAIRE.

Investigation Points	Questions
Application Process	How did you apply the technique regarding its reading order and looking for problems? Why do you think this is the best way to apply the technique?
	How did you apply the technique with respect to the order of using the heuristics? Why do you think this is the best way to apply the technique?
	How would you apply the technique if you were to carry out a new usability evaluation?
Intention to Use the Technique	Would you use this technique in a software development project on your work environment? How?

B. Execution

Each software engineer applied the MIT 2 technique, evaluating the mockup and identifying usability problems. When a software engineer found a usability problem, (s)he described the problem in a worksheet. After this, a researcher interviewed the software engineers and they provided their impressions regarding the MIT 2 technique. The observer provided forms containing some notes. It is important to notice that the observer could question the software engineer’s actions at any time, but (s)he was not allowed to help the software engineer in the discovery activity.

One of the researchers acted as the inspection’s moderator. The moderator was responsible for conducting the study. After the individual inspection by each software engineer, the moderator checked all discrepancies’ worksheets for incorrect

information and gathered the discrepancies. A discrepancy is an issue reported by the software engineer that could be a real defect or a false positive. During this activity, the moderator highlighted duplicated discrepancies.

After this, the discrimination meeting was executed by the moderator and two others researchers (not involved with the study). The purpose of this meeting was to analyze all discrepancies identified by each software engineer. The researchers verified if the discrepancy was a real defect or a false positive. It is worth mentioning that the researchers had high usability knowledge and prior experience in usability evaluations. The quantitative results of the discrimination meeting are presented in Section 5.

Finally, we transcribed the interviews to forms. The interviews allowed this research to gather information in order to understand how software engineers employed the MIT 2. The data analysis of these interviews is presented in Section 6.

V. QUANTITATIVE DATA ANALYSIS

After the discrimination activity, we counted the number of discrepancies, false-positives and defects, the time spent during the inspection, the efficiency and effectiveness per software engineer (see Table III)

TABLE III. SUMMARY OF INSPECTION RESULTS PER SUBJECT

Participants	Discrepancies	False Positive	Defects	Time (Hour)	Efficiency (Defects/Hour)	Effectiveness
P01	9	0	9	0.38	23.48	25.00%
P02	6	0	6	0.58	10.29	16.67%
P03	4	1	3	0.28	10.59	8.33%
P04	8	1	7	0.30	23.33	19.44%
P05	7	0	7	0.45	15.56	19.44%
P06	9	3	6	0.40	15.00	16.67%
P07	9	1	8	0.38	20.87	22.22%
P08	11	3	8	0.43	18.46	22.22%
P09	13	8	5	0.47	10.71	13.89%
P10	13	1	12	0.40	30.00	33.33%
P11	10	4	6	0.62	9.73	16.67%
P12	12	4	8	0.57	14.12	22.22%
P13	12	5	7	0.48	14.48	19.44%
P14	16	6	10	1.02	9.84	27.78%
P15	19	3	16	0.50	32.00	44.44%
Average	10.53	2.67	7.87	0.48	-	21.85%

Overall, the inspection resulted in a set of 36 usability defects, including the 7 seeded ones. Software engineers who used MIT 2 managed to find between 3 and 16 defects spending about 0.28 and 1.02 hours. The effectiveness in this observation study was 21,85%. Comparing this measure with the effectiveness of the group of undergraduate students who used the MIT 2 technique in the first feasibility study (16%) and in the second feasibility study (15,87%), we can notice that this measure was higher in the observation study.

VI. QUALITATIVE DATA ANALYSIS

After the quantitative analysis, we carried out a specific analysis of the qualitative data that were obtained through the comments of software engineers in an interview. These comments provide information such as difficulties and questions during the use of the technique. These issues pointed

us what parts of the technique need improvements. The qualitative analysis was based on the procedures of Grounded Theory (GT) [19].

The qualitative data collected through the interviews were analyzed using a subset of the stages of the coding process suggested by Strauss and Corbin [19] for the GT method: the open coding (1st phase) and axial coding (2nd phase). When analyzing the qualitative data, we created codes (relevant concepts to understand the perception on the technique and its use process) related to the speeches of the participants - open coding (1st phase). After this, the codes were grouped according to their properties, forming concepts that represent categories and subcategories. Finally, these codes were related to each other – axial coding (2nd phase). The goal of the analysis in this study was to understand how software engineers perform the application process of MIT 2. We decided not to elect a core category, because the GT rule is the circularity between the collection and analysis stages until the theoretical saturation is reached [19]. Therefore, the selective coding was not performed (3rd phase of the GT method). The steps of the open and axial coding were enough to understand why some problem occurred and how the inspection process is.

A. Point of View regarding the Application Process of MIT 2

This subsection presents the analysis of how the technique was applied in this study. Through the interviews we identified that the software engineers employed the MIT 2 in three different ways: (i) first, the software engineer read the technique and then looked for problems in the mockup (see quotation from P08 below); or (ii) first, the software engineer looked for the problem in the mockup and then (s)he read the technique (see quotation from P03 below); or (iii) initially the software engineer viewed the mockup, then (s)he read the technique and after this, (s)he changed the way in which the technique was applied, looking for problems as soon as an item was read (see quotation from P10 below).

“I read item by item and tried to find the problems in each mockup” (Participant 8).

“First, I gave a quick look to the mockups and I saw some problems that I knew. Then, I began to read the technique from the beginning” (Participant 3).

“As there are several items in the technique, I looked at the mockup and found some issues, but if I had to look for these issues (...) one by one I think it would take longer. So I preferred to keep these wrong things that I found and (...) I was doing the inspection in the order I had to read each item and found or related to something that (...) I had identified” (Participant 10).

Some opinions were also collected regarding the application of the technique. Some participants noted that when applying the MIT 2, after knowing it, it is better to skip some items (see quotation from P11 below). In addition, other participants said that first seeing the mockup and then reading the MIT 2 technique is not the best way to start the inspection (see quotation from P05 below). However, one of the participants believes that first observing the mockup and then reading the MIT 2, allowed him to think that applying the technique was easy because he knew where some of the problems were (see quotation from P06 below).

“I was skipping; there were even some [items] for which I did not find problems” (Participant 11).

“Looking [at the mockup] first I don’t think it is the best way to start the inspection” (Participant 5).

“During this evaluation I realized that I began to learn what this technique meant and I could look at one element and already know what problem was associated (...). Then this strategy of looking at the mockup and looking at the list can be a starting point for you to memorize the heuristics, but in the future what you realize is that you end up abandoning it and you develop a skill” (Participant 6).

Additionally, this study aimed at obtaining information about how participants would apply the MIT 2 if given the opportunity. Some participants would read the verification item and would already search for problems (see quotation from P03 below). The reasons given by the participants for this way of applying the technique are: if not employed that way (i) the inspection can be more time consuming, (ii) the inspector can forget the problems, (iii) the inspectors does not remember the item is and (iv) because it eliminates primary errors. One of the participants said that first (s)he would view the heuristics and then (s)he would look for the problem and only after (s)he found the problem, (s)he would look for the verification item related to it (see quotation from P14 below). However, other participants said that they would first analyze the mockups and then they would relate the problems encountered with technique (see quotation from P11 below).

“I think reading [the MIT 2] and then looking for the problems in the mockups would be the best way to do it [the inspection]” (Participant 3).

“If I were to carry out a new evaluation, I would not waste time reading it [MIT 2]. If I carried out an evaluation a second time, as I already have prior knowledge, I would look (...) in which of these heuristics (...) [the problem] it is related. But (...) to indicate the verification item, only the second time, checking the item number.” (Participant 14).

“The most appropriate way is to look at the mockups, analyzing it and relating it to the technique” (Participant 11).

B. Opinion about the Intention to Use the MIT 2 in projects

This subsection presents the participants' opinions regarding the use of the MIT 2 technique in software development projects. Some of the opinions were: (s)he would use the MIT 2 early in the project (see quotation from P04 below); and to apply to MIT 2 on a project it will be necessary to explain its advantage, because although there are cost of training and spending time, later there will be gain with the improvements (see quotation from P06 below).

“I would like to try using it [MIT 2] early in the project so I don’t carry it [the evaluation] out in the end” (Participant 4).

“[About the use of the technique in projects] (...) first there should be an explanation, an understanding that it will generate an additional cost to your development process. This cost can be time or human

resources and these impacts on the development of the software. So this cost has to be very well explained to the managers indicating that although you have an initial cost, you have a benefit short after. What is difficult is to convince people that this is important and that organizations can actually recognize such importance and assume that deadlines can be postponed or budgets reduced because of these improvements” (Participant 6).

VII. DISCUSSION

Regarding the Application Process of the MIT 2 investigation point (Subsection VI.A), it can be noted that there were participants who preferred to review the mockup first and then read the technique. When they read a verification item of MIT 2 and remembered a possible usability problem they already observed in the mockup, they related the problem with the verification item. For them, by reading the technique, there was the advantage of already knowing where some of the problems were. These software engineers also stated that it may be a starting point for the use of the technique, because this way of applying the technique also helps memorizing the items. However, other participants said they first observing the mockup and then reading the MIT 2 is not the best way to start the inspection. These participants read a verification item and started searched for the problem. For them, the inspection becomes faster, the software engineer does not forget to point out identified problems and where the verification item related to the problem are.

During the analysis of the study, the researchers noted that the technique helps identifying usability problems in the two ways of applying the technique. In addition, after using the technique for the first time, the software engineers gain prior knowledge of it, and can skip the reading of some heuristics. This way, there is no need to stipulate a prescribed order in the application process of the MIT 2. The software engineers tend to adjust the application of the technique to their own way of use. This allows software engineers to feel more comfortable using the technique according their convenience.

It can be noted that the second investigation point (Subsection VI.B) presented some opinions from the participants related to the use of the MIT 2 in projects. One of the participants indicated that in order to apply the MIT 2 in a project, it would be necessary to explain its advantage, because even though there will be costs in training and time, the software company will have gains with the improvements. For some software engineers, the usability evaluation in the early stages may allow advantages such as: less rework rate and lower costs. This is because the usability problems are identified earlier and repairs are carried out before the coding of the application. Fixing problems earlier is cheaper than correcting problems of something that has already been developed. Through this study with industry practitioners, software companies have evidence of the benefits and opinions of practitioners about early usability evaluation.

VIII. THREATS TO VALIDITY

As in all studies, there are threats that may affect the validity of the results [20]. In this section, we discuss the main

threats to validity of this study. Two main threats were considered that represent a risk for an inappropriate interpretation of the results: (1) training effects and (2) influence of the moderator. There may have been an effect of the training if the training regarding the MIT 2 was different in quality for each software engineer. We controlled the training effects by preparing a single training for all software engineers. Finally, to reduce the second threat, at the discrimination meeting, a team of experts made the analysis of the identified discrepancies, judging if they were usability defects or not, without interference from the moderator.

Three threats were considered regarding the generalization of our findings: (1) the validity of the evaluated artifact as a representative artifact; (2) the researcher inserted some defects in the mockups; and (3) participants with need for training. With regard to issue 1, the inspected mockups are part of the project for a real system. However, it is not possible to say that the mockup used represents all kinds of mockups. With regard to issue 2, all participants found every inserted usability problems. Furthermore, the number of defects found by the participants was much greater than the number of defects inserted by the moderator. With regard to issue 3, the ideal would be that there was no need for training. However, the short training time allows the technique to be used by software engineers with low experience in usability evaluations.

The main threats that may affect the ability to obtain correct conclusions in this study are the size and homogeneity of the sample. These are known problems in Software Engineering studies. Therefore, there are limitations in our results, which are considered indicators and not conclusive.

IX. CONCLUSIONS

This paper described an observational study aimed at eliciting the sequence of activities that is employed by software engineers when applying the MIT 2 technique. Both the qualitative and quantitative results of this study provided us with important feedback to improve the MIT 2 technique.

The qualitative analysis was based on the following investigation points: (1) the application process of MIT 2; and (2) the intention to use MIT 2 in development projects. The qualitative analysis showed that the 2 identified ways of applying MIT 2 in the study proved effective in detecting problems. Through these results, we also noticed that it is not necessary to define a predefined order of applying the MIT 2.

The quantitative analysis showed that the calculated effectiveness in this observational study (21.85%) was higher than the effectiveness measured in the feasibility studies, showing that the improvements made in the MIT 2 previously allowed it to support on the identification of more usability problems. However, other factors may have influenced this outcome, such as: (1) the knowledge increase regarding usability evaluations, (2) and the participants from the observational study were software engineers.

ACKNOWLEDGMENTS

This research is partially funded by the National Science Foundation (grant 1242257, Pan American Software Quality Institute). We would like to acknowledge the financial support granted by CAPES and FAPEAM through processes numbers:

062.00146/2012; 062.00600/2014; 062.00578 /2014; 01135/2011; and PAPE 004/2015. We also thank the participants in the observational study and researchers from USES Group, especially Ana Carolina Oran.

REFERENCES

- [1] A. De Angeli, M. Matera, M. Costabile, F. Garzotto, P. Paolini. "On the Advantages of a Systematic Inspection for Evaluating Hypermedia Usability". In *International Journal of Human-Computer Interaction*, v. 15(3), pp. 315-335, 2003.
- [2] F. Molina, A. Toval. "Integrating usability requirements that can be evaluated in design time into Model Driven Engineering of Web Information Systems". In *Journal Advances in Engineering Software*, v.10 (12), pp. 1306-1317, 2009.
- [3] K. Hornbæk, R. Høegh, M. Pedersen, J. Stage. "Use Case Evaluation (UCE): A Method for Early Usability Evaluation in Software Development". In *International Conference on Human-Computer Interaction*, pp. 578-591, 2007.
- [4] N. Juristo, A. Moreno, M. Sánchez, M. Baranauskas. "A Glass Box Design: Making the Impact of Usability on Software Development Visible". In *Conference on Human-Computer Interaction*, v. 4663, pp. 541-554, 2007.
- [5] G. Donahue. "Usability and the bottom line". In *Journal IEEE Software*, v.18 (1), pp. 31-37, 2001.
- [6] X. Ferré, N. Juristo, H. Windl, L. Constantine. "Usability Basics for Software Developers". In *Journal of IEEE Software*, v.18(1), pp. 22-29, 2001.
- [7] A. Seffah, E. Metzker. "The obstacles and myths of usability and software engineering". In *Communications of the ACM - The Blogosphere*, v. 47(12), pp. 71-76, 2004.
- [8] A. Fernandez, E. Insfran, S. Abrahão. "Usability evaluation methods for the web: A systematic mapping study". In *Information and Software Technology*, v.53 (8), pp. 789-817, 2011.
- [9] N. Valentim, T. Conte. "Technical Report: Version 3 of MIT 2". Report Number 004. Available at: <http://uses.icomp.ufam.edu.br/attachments/article/42/RT-USES-2015-0004.pdf>, 2015.
- [10] S. Propp, G. Buchholz, P. Forbrig. "Integration of usability evaluation and model-based software development". In *Journal Advances in Engineering Software*, v. 40 (12), pp. 1223 - 1230, 2009.
- [11] J. Nielsen. "Heuristic evaluation". In *Usability Inspection Methods* (Eds. Nielsen and Mack), John Wiley & Sons, New York, 1994.
- [12] G. Travassos, F. Shull, J. Carver, V. Basili. "Reading Techniques for OO Design Inspections". University of Maryland, pp. 1 - 56, 2012.
- [13] E. Luna, J. Panach, J. Grigera, G. Rossi, O. Pastor. "Incorporating usability requirements in a test/model-driven web engineering approach". In *Journal of Web Engineering*, v. 9 (2), pp. 132-156, 2010.
- [14] F. Shull, J. Carver, G. Travassos. "An empirical methodology for introducing software processes". In *ACM SIGSOFT Software Engineering Notes*, v. 26, n. 5, pp. 288-296, 2001.
- [15] N. Valentim, K. Oliveira, T. Conte. "Defining an Approach for Usability Inspection in Design Models through Experimentation" (in Portuguese). In *Symposium on Human Factors in Computing Systems*, pp. 165-174, 2012.
- [16] N. Valentim, T. Conte. "Improving a Usability Inspection Technique based on Quantitative and Qualitative Analysis" (in Portuguese). In *Brazilian Symposium on Software Engineering*, pp. 171-180, 2014.
- [17] M. Maguire. "Methods to support human-centred design". In *International Journal of Human Computer Studies*, v.55(4), pp. 587-634, 2001.
- [18] M. Müller, J. Haslwanter, T. Dayton. "Participatory Practices in the Software Lifecycle". In *Handbook of Human-Computer Interaction*, 2nd edition, Elsevier, pp. 255-297, 1997.
- [19] A. Strauss, J. Corbin. "Basics of Qualitative Research. Techniques and Procedures for Developing Grounded Theory". Sage Pub, 400 pg., 2008.
- [20] C. Wöhlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wessl. "Experimentation in software engineering: an introduction". Kluwer Academic Publishers, 2000.

A Empirical Study on the Status of Software Localization in Open Source Projects

Zeyad Alshaikh¹ and Shaikh Mostafa¹ and Xiaoyin Wang¹ and Sen He²

¹ Department of Computer Science, University of Texas at San Antonio, San Antonio, USA

² Department of Computer Science, Harbin Polytechnic University, Harbin, China

Email: {zeyad.alshaikh, shaikh.mostafa xiaoyin.wang}@utsa.edu, he.sen@hpu.edu.cn

Abstract

In modern software development, software localization is a key process to support distribution of software products to the global market. During software localization, developers typically convert all user-visible strings, resource files, and other culture-related elements to the local versions that are well accepted by local users. Despite the popularity of software localization, there have been few studies on the its current status in software practice, such as the proportion of localized projects, the most popular locales, and more importantly, the quality of software localization. In this paper, we present an empirical study on the status of software localization in open source projects. We find from that, popularity of software localization varies a lot in different User Interface (UI) frameworks and domains. Furthermore, we surprisingly find that only about 60% of string keys are actually translated on average in localized top software projects and software localization often span a long period of time in the software development history.

1 Introduction

In this era of globalization, most software applications have potential users from different regions of the world. A typical process to develop multiple local versions of a software application includes two steps: an internationalization step in which developers externalize all region-specific code elements (e.g., user-visible strings, measures, date formats, writing-direction-specific inputs, and sometimes also laws and policies) to resource files, and a localization step in which software localizers transform the original resource files to local resource files for certain locales [5].

Despite of the popularity of software localization,

there has been few research efforts to study the current status of software localization in practice, and the major challenges faced in software localization. In this paper, we present an empirical study on open source Java software projects in SourceForge [1]. Specifically, we studied the factors that affect where a project is localized, the popular languages that projects are localized to, and the quality of software localization in the localized projects. Our major findings are as follows.

- Software Localization is widely used in open source projects, but the popularity of software localization varies a lot for different software domains and UI frameworks.
- The quality of software localization in top open source projects are relatively low.
- The localization of a software project often spans a long period of time during software evolution.

2 Study Design

In our study, we plan to answer the following research questions.

- **RQ1:** How popular is software localization in open source software projects?
- **RQ2:** What is the quality of software localization in open source software projects?
- **RQ3:** How long does it take for a software project to finish its software localization?

In our empirical study, we used two subject sets, both are downloaded from Sourceforge. We choose software projects from Sourceforge as subjects for the following two reasons. First, Sourceforge is a popular software repository with a large number of open source

software projects and long history. Second, Sourceforge provides the statistics of software downloads during different time period and from different countries.

Specifically, we built a *random subject set* including 2,500 randomly selected software projects which are active in 6 months and whose weekly downloads is larger than 10. Also, we built a *top subject set* which includes 10 software projects among the top 100 projects, which are from different software domains, and whose property files are in standard “key=value” format (so that it is possible to partially automate our data extraction in our study). To answer **RQ1**, we studied the random subject set due to its large size and representativeness. To answer **RQ2** and **RQ3**, we studied only the top subject set, because some manual in-depth inspection of resource files and version history are required.

To perform our empirical study, we extracted the following information from each subject software project. First of all, from the web site of each software project in our random subject set, we extracted the relevant meta data including supported locales, weekly downloads, domains, and used UI frameworks. Second, to answer **RQ2**, and **RQ3**, we manually identified the local resource files for each project in the top subject set, and extracted the file content and version history.

3 Study Results

In this section, we present and discuss the results of our empirical study.

3.1 Popularity of Software Localization

In our study, we find that, the proportion of localized software projects in the random subject set is 38.0%. To understand the factors that may affect software localization, We further studied our random subject set to find out how the proportion localized software varies with different number of weekly downloads, UI frameworks, and domains. The results are shown in Figure 1 through Figure 3.1. Since there are too many different UI frameworks and domains, we present only the results for the 5 UI frameworks / domains that are most popular in our subjects (the 5 labels on the left), with highest localization popularity (the 5 labels in the middle), and lowest localization popularity (the 5 labels on the right). It should be noted that a UI framework / domain may belong to two categories of above, so they may appear twice as a label in a figure. Also, for representativeness, we consider only UI frameworks / domains with more than 10 projects using them.

From the figures, we have the following observations.

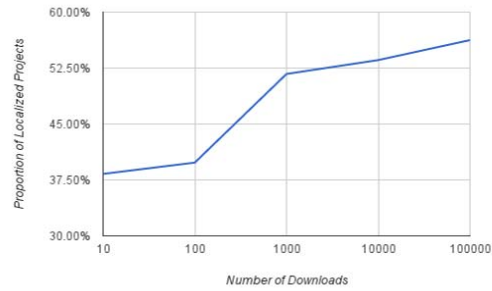


Figure 1. Localization Popularity among Projects with Different Downloads

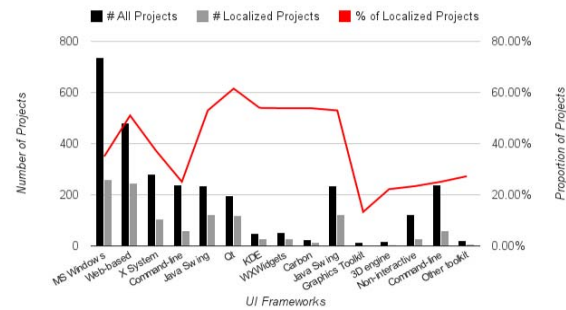


Figure 2. Localization Popularity among Projects with Different UIs

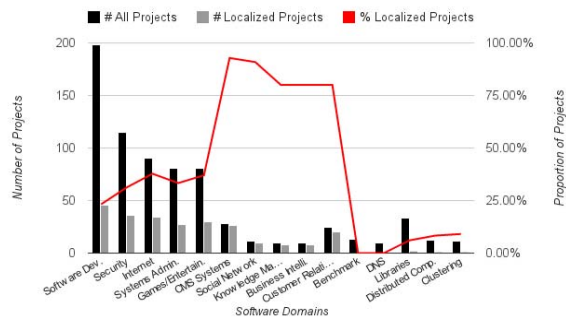


Figure 3. Localization Popularity among Software Projects in Different Domains

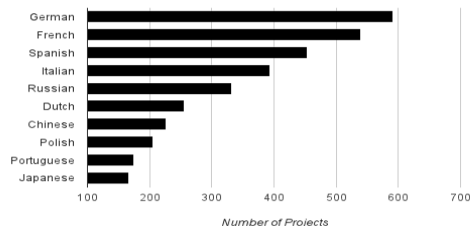


Figure 4. Top Locales in Software Localization

First of all, among the software projects with higher weekly downloads, the proportion of localized software projects is higher.

Second, among the most popular UI frameworks (left 5 labels in Figure 2), projects that are web-based or Java-Swing-based are more likely to be localized (with a localization proportion higher than 50 %), while command-line-based projects are less likely to be localized. The results are reasonable, because web-based applications are typically accessible from users all over the world, and Java-Swing-based applications are light-weight and easy to be distributed.

Fourth, among the most popular 5 software domains, Internet applications, System administration applications, and game applications are relatively more likely to be localized, perhaps because they are designed for a larger variety of end users.

Finally, we also studied the popularity of locales, and the top 10 most popular locales are presented in Figure 4. In the rest of our study, we will focus on these locales.

3.2 Software Localization Quality

We studied the quality of software of localization by checking the proportion of key-value pairs that are actually translated in the localization. Our study reveals that software developers often perform partial localizations (maybe due to the lack of time / effort or proper translator). In such cases, an untranslated string either appears in their default language version in the locale property file, or the pairs of keys and translated strings are simply omitted from the local property file (the software will refer to the default locale property file when it cannot find a key).

We present the results of our study in Figure 5. Figure 5 shows the average percentage of translated keys



Figure 5. Localization Quality for Different Locales

for a certain locale among all projects in the top subject set.

3.3 Software Localization Process

To further understand the software localization process in practice, we studied the version history of resource files to calculate the time span of the software localization process. In Figure 6, we present the breakdown of all local resource files on the time difference between its commit, and the commit of the default resource file.

From the figure, we can observe that, about 32% of all local resource files were committed more than 6 months after the default resource file was committed. This indicates that the software localization process often takes a long time. Actually, it is unlikely that the developers were working on the localization during this period of time. A more possible reason is that, the developers failed to find a proper translator / contributor to perform the localization for certain locales. Also, we observe that there are 4% of resource files were committed before their corresponding default resource files were committed. The reason is that, some modules of some projects were initially written with a locale different than English, and the developers later added an English resource file, and reset the English resource file as default.

4 Related Works

The most related work to our study is a recent study [2] on the Android Framework. This previous work studies the version history of the Android project, and reported some findings on the quality and code commit frequency on software localization. Compared to their study, we used a much larger set of software

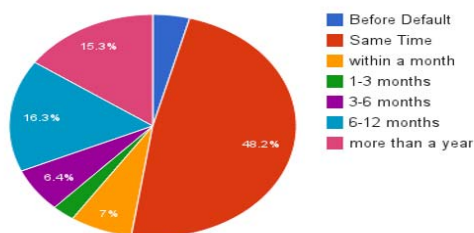


Figure 6. Break Down of Resource Files based on the Commit Delay

projects, and provide more detailed results on the popularity and quality of software localization.

In academia, researchers have summarized a number of important research issues in software internationalization and localization [6, 3], including architectural practice, extraction of region-specific code elements, management of strings in resource files, translation, and cultural adaptation. These issues have been studied by various research efforts [4, 7, 9, 8, 10].

5 Conclusions

In this paper, we present a study about the software localization status on open source software projects from SourceForge. In our study, we find that, the popularity of software localization is higher in software projects with more downloads, and varies for different UI frameworks / domains. Furthermore, we find that the quality of software localization in open source projects is relatively low and the localization process often span a long period of time.

Our findings mainly call for two future research directions. First of all, the current study shows low quality as well as high time cost (or the difficulty to find proper contributor) for software localization. So automatic support for software localization is highly desired. Second, it is surprising that some top software projects have a low quality in software localization. So it would be interesting to perform a more in-depth research on the impact of software localization on the success of a software project on the global market.

Acknowledgment

The authors from University of Texas at San Antonio are supported in part by NSF grant CCF-1464425, and DHS grant DHS-14-ST-062-001.

References

- [1] Sourceforge, <http://sourceforge.net/>.
- [2] L. Arjona Reina and G. Robles. Mining for localization in android. In *Proceedings of International Working Conference on Mining Software Repositories*, pages 136–139, 2012.
- [3] V. Dagiene and R. Laucius. Internationalization of open source software: framework and some issues. In *2nd International Conference on Information Technology: Research and Education*, pages 204–207, 2004.
- [4] A. Danko. Formalization of functional aspects in business software globalization. In *14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, pages 107–116, 2010.
- [5] B. Esselink. *A Practical Guide to Software Localization: For Translators, Engineers and Project Managers*. John Benjamins Publishing Co, 2000.
- [6] J. H. Hogan, C. Ho-Stuart, and B. Pham. Current issues in software internationalisation. In *Proc. Australian Computer Science Conference*, pages 1–10, 2003.
- [7] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun. Locating need-to-translate constant strings for software internationalization. In *International Conference on Software Engineering*, pages 353–363, 2009.
- [8] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun. Transtrl: An automatic need-to-translate string locator for software internationalization. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 555–558, 2009.
- [9] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun. Locating need-to-translate constant strings in web applications. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 87–96, 2010.
- [10] X. Xia, D. Lo, F. Zhu, X. Wang, and B. Zhou. Software internationalization and localization: An industrial experience. In *18th International Conference on Engineering of Complex Computer Systems*, pages 222–231, 2013.

products to the customers. *Pre-Analysis* is divided into two tasks: *Market Analysis* (evaluates the market regarding its business goals, user profile, legal and cultural constraints, business opportunities, competitors, and other factors that the *Business expert* and *Domain expert* define) and *Marketing Strategies Identification* (defines how the products will be released to the customers considering the sale units, integration methods, installation, product maintenance, and user skill level).

B. Product Map Development

The *Product Map Development* consists of tasks to build a matrix comprising features and products, indicating, for each product, which features it implements. Kang et al. [4] define feature as “a prominent user visible aspect, quality, or characteristic of a software system or systems”. As result, there is a common understanding and sharing of the major features and products. This activity comprises four tasks: *Products Identification* (identifies the products available in the company portfolio that share business activities in common) and *Major Features Identification* (identifies major features and provides a description of them), *Features Grouping* (arranges the fine-grained feature into major features) and *Product Map Building* (*Domain Analysis Expert* builds a map indicating, for each product, which major features it implements).

C. Major Features Prioritization

This activity prioritizes the major features using classification criteria relevant for reuse. *Major Features Prioritization* is divided into two tasks: *Domain Potential Assessment* (assessment is performed by applying on *Domain expert* a questionnaire using a set of criteria, such as domain maturity, reuse potential and risks) and *Major Features Ranking* (defines the final ranking of major features based on assessment previously applied).

D. Sub-Features Definition

Sub-features express the details of major features. The *Sub-Features Definition* specifies descriptions and properties of the sub-features for each major feature in the domain. Since the major features involve a considerable number of functionalities, an iterative process is necessary to optimize the company effort. This activity comprises four tasks: *Sub-Features Identification* (*Domain Analysis Expert* captures reusable sub-features from the *Domain Expert* or *Product Expert* through workshops), *Legacy Assets Mining* (*Domain Analysis Expert* captures reusable sub-features by studying the system-as-is, their flows, work procedures, business rules, reports about defects, user manuals, screenshots of the products, and prototypes), *Sub-Features Inspection* (*Inspector* inspects the features list in terms of non-conformities that consider aspects such as feature granularity, understanding, and duplication), and *Sub-Features Validation* (*Domain Analysis Expert* collects opinions from the *Domain Expert* and *Product Expert* to adapt the features specification whether necessary, and integrate them with the other available features).

E. Commonality and Variability Analysis

This activity specifies the commonalities and variabilities through feature model and product map work products. It

defines which features are directly linked (father-son) and how they are classified (mandatory, optional, and variant). Three tasks support the *Commonality and Variability Analysis*: *Product Map Updating* (updates the product map created during the *Product Map Development* and *Sub-Features Definition* activities), *Feature Modeling* (organizes hierarchically the feature model [4] based on the features list, commonalities and variabilities provided by the *Domain Expert*, *Product Expert* or *Legacy Assets Expert*), and *Models Inspection* (verifies the *Product Map* and *Feature Model* inconsistencies).

F. Agile Practices

For the proposed process, the activities *Sub-features Definition* and *Commonality and Variability Analysis* incorporates certain practices, artifacts, and roles of the Scrum framework. These practices are: *Sprint Planning*, *Regular Meeting* and *Sprint Review and Retrospective*.

Sprint Planning: aims to find the most appropriate major features for a specific sprint that provide reuse potential for the organization. Based on the Scrum framework, the planning is divided into two parts: part one, where the major features are selected (from the domain backlog) for the next sprint (sprint backlog); and part two, where the major features selected are broken into Scrum tasks (in the sprint backlog).

Regular Meeting: adapted from the daily meeting from Scrum, the *Regular Meeting* is a practice that intends to aid the stakeholders with fast feedbacks about the sprint. The frequency of this meeting is defined according to the effort spent in the domain analysis activities (such as daily or every two days). Issues related to the process, artifacts and team are raised by the *Scrum Team*. From this meeting, some small adjustments can be performed during the sprint, since the sprint goal does not change.

Sprint Review and Retrospective: after finishing the sprint, the *Sprint Retrospective* and *Sprint Review* practices adjust and improve what is necessary for the next sprints (e.g. process and team adjustments). An initial cause analysis also is performed to explore the root of the issues during the process. In addition, the team should provide a new effort estimation for each major feature in the scope backlog in order to be more precise when performing the *Sprint Planning* part one. The *Domain Analysis Expert*, *Domain Engineer*, and the *Legacy Systems Engineer* would consider the technical issues to perform these new estimation. Technical issues can be associated to the variability implementation complexity, structuring the common and variable components or using new technologies. The *Scrum Team*, for instance, would estimate how long it will take to perform the *Commonality and Variability Analysis* for the major features.

III. CASE STUDIES

We evaluated the RiSE-DA process qualitatively, through the case study technique [5]. Our assumption is that the proposed domain analysis process achieves acceptable results in terms of work products quality and applicability in the organizations. However, companies moving to agile principles and practices are looking for software reuse processes providing mechanisms to support faster changes in volatile business

domains. In this context, process iterativeness, feedback among stakeholders, and process adaptability are essential [1].

In order to evaluate iterativeness, feedback among stakeholders, and process adaptability in our proposal, we defined the research question “*How do the stakeholders characterize the iterativeness, adaptability and feedback of the process?*”. In order to enable the triangulation of information [5], the data collection procedure involved the following techniques: survey, field observation, document analysis, and focus group.

A. Case #1: An Oil, Gas and Energy Company

The *Case #1* was applied in an Oil, Gas and Energy company, working with software development for more than thirty years and is spread over several cities, whose main locations are Rio de Janeiro and Salvador, in Brazil. In this project, a set of applications was selected in a pilot.

A training was applied to the company employees (project participants). Presentations were prepared by the RiSE members and company members were grouped by availability. It was a training of eighteen hours divided into three days. Support materials and practical examples were used to provide a better learning.

Despite the company size, the pilot was performed with a small team, considering time restrictions for many employees participate in this project. Then, the roles were played by more than one member, i.e., the *Domain Expert* and *Business Expert* roles were performed by the same participant.

The resulting product map and feature model described three different domains, 17 major features and 96 sub-features. It took three weeks with three sprints to describe the features. Time variations occurred in the three performed sprints, with an average of four hours and thirty minutes per sprint. Twenty percent of the time (forty minutes) was spent with the agile management tasks. The company participants did not provide details about the features and product functionalities, since they were not business experts. It might have influenced the spent time.

1) *Case #1 Findings*: RiSE-DA fostered the iterativeness only when defining sub-features and analyzing commonalities and variabilities. There was not change for the sprint time box. The company participants argued RiSE-DA fostered the iterativeness in the project, since the sprints were short and the domain was incrementally defined.

Adaptability was also fostered when defining sub-features as well as analyzing commonalities and variabilities. The Scrum practices (retrospective, planning, review, and daily meeting) detected needs of changes in the process activities and artifacts.

Furthermore, RiSE-DA fostered continuous feedback, when defining sub-features and analyzing commonalities and variabilities. Workshops and model storming were good strategies to achieve it. The feedback among stakeholders were effective, and iterativeness provided a considerable impact on feedbacks.

Therefore, all evaluation aspects had a significant impact. Frequent feedbacks and small (but frequent) changes were possible because of the iterativeness. Feedbacks supported the

changes detection, providing insights for next sprints, and adaptability supported process adjustments such as changes on the daily meeting practice (daily meeting questions changed since the *Scrum Team* worked together in many sprints).

For each new sprint with a different group of company members, the *Scrum Team* analyzed previous models and evolved them, then, obsolete features were detected and removed for next sprints. The decision of removing obsolete features occurred after the *Scrum Team* evaluate the retrospective and review results, during the sprint planning, and since the features were in the target domain of the current sprint.

Iterativeness and feedback enabled the *Scrum Team* to anticipate problems regarding the requirements, technical constraints, staff, or other external factors such as demands for product development. Adaptability had an important impact on the process performance when the adjustments on activities optimized the way as the company performed them.

B. Case #2: An Educational Management Systems Company

The Educational Management Systems company works with software development for the educational/scholar domain for eighteen years and is located at Salvador, Bahia, Brazil. A domain analysis project was set up to organize the customizations as derived products from a common reuse platform. Seven participants (four from the company) participated in the project.

The company members were trained before the domain analysis. Basically, the domain analysis concepts were communicated, then the case participants applied them in the company context.

As results, product map and feature model described one domain, 8 major features, and 159 sub-features (from two of the major features). The company participants were involved in other demands and the project took about six months (the amount of features also influenced the spent time). The meetings duration varied in the eleven sprints, with an average time of fifteen hours for each one. Sixteen percent of the time, approximately, was spent on sprint planning, regular meetings, retrospective, and review.

1) *Case #2 Findings*: RiSE-DA fostered iterativeness in the *Sub-Features Definition* and *Commonality and Variability Analysis* activities. There were changes in the sprint time box during the project due to external factors such as the legacy systems maintenance. Defects in the software caused the variations in time boxes. The company participants considered the iterativeness fostered flexibility in the project in terms of changes and reflections.

In addition, the process fostered adaptability, mainly when defining sub-features and analyzing commonalities and variabilities. The Scrum practices (retrospective, planning, review, and daily meeting) detected needs of changes in the process activities and artifacts. The participants argue the process adaptability supported the flexibility in the project, and the adaptability was frequent because of the iterativeness.

RiSE-DA also supported continuous feedback, mainly when defining sub-features and analyzing commonalities and variabilities. Workshop was a good strategy to achieve it.

Although the *Scrum Team* members were in different locations (they used tools for remote communication), the feedback among them and other team members was effective. Scrum and model storming practices supported the feedbacks. The feedbacks were continuous due to iterativeness.

Frequent feedbacks and small (but frequent) changes were possible because of the iterativeness. Feedbacks detected changes and provided it for the next iteration. Adaptability encouraged adjustments to improve feedback (e.g. the daily meeting questions changed to a simpler format, since the *Scrum Team* worked together in many sprints).

IV. LESSONS LEARNED

In order to report the experience from the application of RiSE-DA in real-world scenarios, we have present the lessons learned as follows:

Duration of RiSE-DA Sprints. Sprints with one or two weeks were considered appropriate, because the team was learning how to perform domain analysis and understand the domain (both companies). This lesson learned favor the iterativeness (maximum of two weeks), frequent feedbacks and adaptability.

Presence of the *Domain Expert* in workshops. The *Domain Expert* has deep knowledge about the domain and he can provide frequent feedbacks. This lesson enables the identification of some obsolete features during the sprints and the changes were made earlier.

Presence of *Business Expert*. The *Business Expert* aligns business goals and market strategies of companies. In the larger company, we faced the absence of business experts and it impacted on the process results. The involved employees were from the software development sector, and provided information based the product development they are responsible for. They did not have an overall knowledge on the domain. As consequences, they provided fine-grained features, revealed difficulties to identify major features and relationships among features implemented by different products. To deal with these consequences, we combined the participants' information with legacy systems analysis.

Early changes detection. Changes detection impacted on the performance, effort, obsolete features control, risks management, and problems during the project. As a final result, the motivation in the projects was considered beyond expectations.

Deal with complex organizational structure. The complex organizational structure of the larger company influenced the results, since important products could not be included in the case, and important experts were not available. The process adaptability allowed the participants to build comprehensive domain models considering only the available information.

Feature identification. Feature was a new concept for most participants. During the initial activities, they had problems by defining the products features. We adopted two strategies to mitigate this problem: for the small company, we presented practical examples of features and investigated the functionality description provided by the participants in order to verify whether it was a feature or not; for the larger company, the

participants used the activity diagram notations during model storming practices, and the domain analysis experts extracted the candidate features from these diagrams.

Sprint scope. During the sprints, participants proposed to add activities that meet other companies' objectives (e.g. combining the domain analysis process with the development of a framework). This focus shift impacted on the process results in terms of time and effort. Therefore, as lesson learned, the process must focus on defined activities and possible adaptations, without adding different ones.

Tool support for domain analysis. The available commercial tools to model domain variability provide more functionalities than needed for domain analysis, since they are expensive. Thus, it was not feasible use these tools in the context of this work. We used an unstable academic tool which caused problems during the project such as missing information. In addition, they do not address traceability among product map and feature model. It still remains an open problem in our research work.

V. CONCLUSION

In this paper we described activities, tasks, roles, and artifacts of RiSE-DA, a lightweight domain analysis process adapted to the Brazilian software development scenario. According to our evaluation, RiSE-DA provides appropriate domain analysis activities considering the evaluated cases.

In addition, the stakeholders manifested motivation with management aspects (iterativeness, feedback, and adaptability) of the process. Scrum practices helped domain analysis activities to find obsolete features and make changes in a faster way, decreasing effort and increasing the motivation to use the process.

Although the qualitative studies are hard to replicate and generalize, in future work, new evaluations should be performed in different scenarios to reinforce the findings in this study. We also intend to define new activities for the process (e.g. requirements, architecture, and testing) in order to build a complete domain engineering process.

REFERENCES

- [1] I. F. da Silva, P. A. da Mota Silveira Neto, P. O'Leary, E. S. de Almeida, and S. R. de Lemos Meira, "Agile software product lines: A systematic mapping study," *Software Practice and Experience*, vol. 41, no. 8, pp. 899–920, Jul. 2011.
- [2] K. Schmid, "A comprehensive product line scoping approach and its validation," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 593–603. [Online]. Available: <http://doi.acm.org/10.1145/581339.581415>
- [3] J. Bosch and P. M. Bosch-Sijtsema, "Introducing agile customer-centered development in a legacy software product line," *Software: Practice and Experience*, vol. 41, no. 8, pp. 871–882, 2011.
- [4] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.
- [5] R. K. Yin, *Case Study Research: Design and Methods (Applied Social Research Methods)*, 4th ed. Sage Publications, 2008.

A Feature-Based Tool-Selection Classification for Agile Software Development

Mohsen Taheri

School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
mtahe006@fiu.edu

S. Masoud Sadjadi

School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
sadjadi@cs.fiu.edu

Abstract—With the advancement in technology, software development complexities are rising across the globe. This trend is forcing companies and organizations to adopt management methods and tools to accelerate time to market, more easily manage changing priorities, increase the customer satisfaction and reduce product expenses. Agile software development methods offer a solution to these issues, but problems remain over evaluation along with the offering of the correct agile software as well as a collection of agile tools. The purpose of this paper is to introduce best tools and features, criteria used for evaluating currently existing tools and propose a classification model to right agile tool selection. To prepare a list of the best tools and their features in the market, a practical research on existing tools and their features were performed. Finally, a classification model was prepared and the results show which tools best fit into different level of maturity in projects and companies.

Keywords-*software development; agile tools; agile tool selection; feature-based classification.*

I. INTRODUCTION

Agile software development is a set of software development methodologies based on incremental and iterative development in which specifications and alternatives, develop by means of cooperation between self-organizing, cross-functional groups. It promotes adaptive arranging, evolutionary improvement, early delivery, ongoing enhancement, and encourages rapid and accommodating response to change. In recent years, many startups, software companies and organizations adopting agile development methodology. They want to develop fast and high quality software products. Also, some other benefits obtained from implementing agile consist of the ability to deal with the software development visibility, cost, risk and priority management, to improve team moral and to make simpler project implementation process. This research is based on standard agile definitions and concepts and uses agile principles and agile manifesto to review the tools and their features. According to the agile manifesto, “individuals and interactions are over processes and tools, working software is over comprehensive documentation, customer collaboration is over contract negotiation and responding to change is over following a plan”.

The market industry regarding software agile tools is now becoming more mature with commercial tools and dozens of small and large vendors which guide you to learn and work with

agile methodology. Sometimes companies make mistake to choose appropriate tool, therefore many corporations arise three questions in their mind. First of all, which agile tools in the market is the best? Secondly, which agile tool is the best for our organization? And last but not least, how to select the right tool? Thus does a special agile tool fully meet all company expectations as a "one size fits all" tool for a product team, and make their collaboration and project tracking overall enjoyable.

Although there are many apps and tools offering traditional project management, tasks management & To-Do List planning, this survey focuses only on agile project management tools, their specification and a classification to select best and right agile tool for each organization. This paper is organized as follows: Section 2 presents the literature review and previous works. Section 3 presents the methodology and research steps conducted during the study, agile tools, and describes the criteria used for evaluating currently existing tools. Section 4 analyses the lists and Section 5 presents the tool evaluation results. Finally, Section 6 concludes with final remarks [1, 2].

II. LITERATURE REVIEW & RELATED WORKS

For this step, we analyzed the present white papers, journal and conference papers and best tool usage surveys in the agile development context. We went through many most important world's largest scientific and educational sources such as IEEE, ACM, Springer, Google scholar, and etc. We even surfed through less scientific online sources such as websites, whitepapers and published surveys. Finally, we only found few different surveys, which some of them were sponsored by tool vendors themselves. Some of the most relevant works to our research are presented as follows.

In 2011 Azizyan provides a list of features that are most desired by the existing software companies. Its result shows that the most satisfactory tool attribute is ease of use. As a positive point it is an unbiased survey and the negative point is it has focused just on gathering statistics as other surveys. Another negative point is the use of spreadsheets, yahoo groups, and the like to collect information using questionnaires. Although they had an IP tracker that it isn't reliable to have a normal distribution of countries, people and companies. This paper helps us to prepare a list of tools, criteria and metrics for our tables [3].

In 2012 Azizyan presents a journey towards agile tool selection for a specific anonymous company and the tool selection process is based on a study of the tool no functional features such as flexibility and usability. This paper gives a brief description of the company, then another section lists and describes the metrics used for evaluating currently existing tools. It has focused just on a special company and few tools, but in comparison with other papers, it introduces a methodology to select the right tool [4].

In 2006 “Agile Project Management (APM), Tooling Survey Results” focused on collecting statistics on tools used in requirements management, and also there are some statistics on agile method used and reasons for selecting an agile project management tool. It helps us to prepare a list of tools, criteria and metrics for our tables [5].

In 2008 “Agile tools: the good, the bad, and the ugly” mainly focused on tools used in agile projects. It focused on gathering statistics on company structure and maturity of agile methods using TargetProcess trial versions. Although the paper has published a couple years ago and in recent years, many new tools have captured the market, it is beneficial as a reference to choose most important tools and metrics [6].

In 2013, “8th Annual State of Agile,” written by the VersionOne Company includes a normalized and wide distribution of responses of multitude of channels from companies, engineers, scrum masters, product owners and even self-employed engineers. The respondents are from different countries and questions have focused on details such as reasons for adopting Agile, agile techniques used. The main points of the paper are detailed statistics in the agile methods in projects, and the information about adopting agile methods [7].

In 2014, “Agile Tools Evaluator Guide” written by the VersionOne Company intended to help organizations in choosing software to support their agile teams and processes.

Some of the mentioned research is considerable due to their direct relevance to our research problem. Some of them are sponsored by vendors, therefore the questions probably have been prepared based on the product features of the company. None of the papers and surveys provide comprehensive method and opportunity to select a tool among a wide range of agile tools. Also, none of them provide a methodology for right tool selection regarding the size and maturity of projects and companies for instance for a small startup or large organization.

III. METHODOLOGY

Companies that are successful in agile software development know that "Individuals and Interactions" are more important than "Processes and Tools"; but the right agile tools really can affect the enterprise, especially when interactions can be more productive. Thus, how top agile tools are provided and which important factors are essential, are discussed in this section.

A. Tools

To prepare a list of the best tools in the market, firstly, more than 300 blogs, web pages, including reviews, tutorials and online books have been read. Afterward, we reviewed papers, surveys, and white papers, especially those which had been published in recent years. Secondly, more than 40 graduate

students, including 10 PhD students from the computer science department of Florida International University (FIU) during a semester were supposed to choose one or two tools and make a practical research in the enterprise or project using chosen tools and finally they made a video describing tool and their features. Some of them are still adopting agile tools in self-employed projects, senior projects, startups, and even organizations and they provide us precise feedbacks. Also, they installed tools, paid if necessary, and released their results on Github and YouTube [8].

There are different types of management tools. Traditional Project Management Tools, Spreadsheets, Physical Walls and Paper, and commercial modern Agile Project Management Tools. Other than physical tools we consider features like the size of the project, the size of the team, stability of the requirements and complexity of the software for a wide range of available tools to maintain diversity among them. Afterwards the tools are divided into Proprietary tools vs. open source tools. To keep a better comparison, we consider some of the criteria as well like satisfactory aspects of the tools like:

- Ease of Use
- Integration with Other Systems
- Availability of Reports
- Price
- Customizability

B. Criteria to consider

To prepare a list of the most important criteria to satisfy agile techniques employed, all of the recent surveys were considered. In addition, some feedbacks provided by students helped us to balance some of the vendor’s surveys. Six different core criteria definition is presented as follows.

1) Flexibility

Organizations and companies are different and unique. The agile project management tool should have flexibility to adapt to those differences.

2) Ease of Use

Ease of use is that users can utilize the agile tool without a lot of training and time consuming procedures.

3) Category

Companies are placed into the categories that fit with their organization’s needs. For example, if it is an organization with a hundred users, it is probably not going to want a simple standalone solution.

4) Pricing

Pricing and cost models are an important factor in any purchase and agile project management tool.

5) Responsiveness

How responsive are the vendors? How do the vendors support their customers? Responsiveness is how the vendors respond to the needs of their customers.

6) Features

Features are an essential part of any agile tool evaluation. After you figure out which specifications and features a system supports, understand how those features would be used to perform your project process.

7) *Open source tools*

Agile project management are divided to proprietary tools and Open Source Tools. Open source agile tools may have some restriction while using some features; thus each organization should consider its situation before choosing a tool. Particular features may be vital for one enterprise whereas is not important for another company. The following factors are considerable using open source tools.

- Feature sets
- Usability
- Viability
- Suitability for large companies, projects and products

IV. RESULTS

The comparison table “Table I, II” is used to compare best agile project management tools. The purpose of the comparison table is to highlight the requirements for which you are looking, and to be able to compare different agile tools against those

requirements. Some of the most important key factors should be considered in order to select an agile tool for project management.

A. *Life Cycle Management using One Agile Tool*

Storing project information in different multiple tools causes inaccurate results and prevents to comfortable real-time visibility.

B. *Cross-Functional Teams*

It means to manage the requirements of the customers, programmers, testers, product owners, and other stakeholders in an integrated environment to enhance collaboration and consistency.

C. *Enterprise Scale*

In order to deployment of an enterprise, agile tools should be able to handle the project structure, tasks, defects and tests.

Table I: Evaluation criteria

Lifecycle Coverage	Product and their release. Iteration planning and its tracking, Strategic Goals, backlog and the repository for defects, Test management
Simplicity & Ease of Use	Customizable dashboards for tracking, Drag and Drop; Shortcut options for actions such as: Close, open and delete; Interactive environment supporting the daily activities of teams
Collaboration	Communication media for teams; Mobile Stream to keep projects moving; Email notifications and RSS feeds; Reporting and tracking for distributed team members; Customizable boards and coding
Analytics, Visibility and Reporting	Dashboards with sufficient metrics; Advanced planning e.g. what-if analysis; Reports, charts and graphs; Hierarchy charts, Relationship mapping, Release dependency visibility
Workspace and Process	Drag and drop story, task and boards; Customizable methodologies (XP, Scrum, Kanban, etc.); Extensive options for boards, fields; Color coded visual representation
Program Management	Release rollouts; Program-level Epicboards; Epic planning; Cross- team planning, tracking
Deployment, Integrity and Security	Free trial software available; Maturity size-based product versions; Web services API; Project-level security; Integrates with Existing Tools like Source control systems (e.g. GitHub), bugtrackers (e.g. JIRA).

Table II: Agile tools comparison chart (A: Full support, B: Quite good, C: Bad, *: Free applicable trial)

	Commercial modern Agile tools																Traditional & Spreadsheets								
	Proprietary tools														Open source tools		Excel	Microsoft Project	Google Docs						
	Atlassian Jira/Greenhopper	Axosoft OnTime	Target Process	Microsoft TFS	Rally Platform	Mingle	Version One	Blossom.io	Scrumwise	Base Camp	LeanKit	AgileZen	PlanBox	Kanbanize	ScrumWorksPro	Banana Scrum				AgileFant	IceScrum	XPlanner	Trello	Asana	Agilo
Lifecycle Coverage	A	A	A	A	A	B	A	A	B	B	B	B	B	A	B	B	A	B	B	B	C	B	C	C	B
Simplicity & Ease of Use	B	B	A	B	B	A	B	B	B	B	A	B	B	B	B	B	B	B	B	A	A	B	A	A	B
Collaboration	A	A	B	B	B	A	A	A	B	B	B	A	A	A	B	A	A	C	A	B	B	C	C	C	B
Analytics, and Reporting	A	A	B	B	A	B	A	A	B	B	C	B	B	B	C	B	B	B	C	A	C	B	B	B	C
Workspace and Process	B	B	A	B	B	A	B	B	A	B	B	A	A	A	B	A	C	C	A	B	B	B	B	C	C
Program Management	A	A	A	B	A	B	A	A	B	B	C	B	C	C	B	C	B	C	C	B	B	C	B	C	B
Deployment, Integrity and Security	A	A	A	B	B	A	B	B	A	B	B	A	B	B	B	B	C	B	C	A	C	B	C	C	C
Free plan	*	*		*	*		*		*	*	*			*	*		*	*	*	*	*	*	*	*	*
Scrum&Kanban Supported	A	A	B	A	A	A	A	A	B	C	B	A	B	C	A	B	B	A	B	B	B	C	C	C	C
Popularity on the web	A	A	A	B	C	A	A	C	C	A	A	A	B	A	A	C	A	C	B	A	C	C	C	A	B

V. CLASSIFICATION

The comparison tables focus only on top 25 agile project management tools and compare them; but there are also a lot of commercial vendors offering solutions in this market. How to select right agile project management tool for different maturity level? Agile only fits in some company scales and the sad truth is that agile doesn't fit all company scales. So, many agile adoptions in progress right now are going to fail for this reason "Table III".

- Start-up: During this time manager usually struggles to survive.
- Growth stage: Company has added customers and increased sales to new markets and also new professional staff must be added.
- Maturity stage: The business is operating well, with an established market share "Fig. 1" [9, 10].

Table III: Ease of Use for Different levels of Maturity (A: Quite appropriate, B: good, C: Bad)

	Commercial modern Agile tools																				Traditional & Spreadsheets			
	Proprietary tools															Open source tools								
	Atlassian Jira,Greenhopper	Axosoft: On Time	Target Process	Microsoft TFS	Rally Platform	Mingle	Version One	Blossom.io	Scrumwise	Base Camp	LeanKit	AgileZen	PlanBox	Kanbanize	ScrumWorksPro	Banana Scrum	AgileFant	IceScrum	XPlanner	Trello	Asana	XPStoryStudio	Excel	Microsoft Project
Senior Projects & Self-employed	C	B	B	B	B	B	B	A	B	A	A	B	A	B	A	B	A	A	A	A	B	A	A	B
Start Up	B	A	B	A	A	B	A	B	A	B	A	B	A	B	B	B	A	B	B	B	B	B	B	C
Growth	A	A	A	B	B	A	A	B	B	B	B	B	C	A	C	B	A	A	B	C	A	C	C	C
Mature	A	A	A	B	A	A	A	B	B	C	B	C	C	C	C	C	C	C	C	C	C	C	C	C

VI. CONCLUSION

As project team members in the company continue to use agile and enterprise scales agile development within their companies, the challenges of managing different groups continue to increase. Agile software development tools provide solutions to manage this sophisticated process using a framework to maximize the consistency and success of agile development. In this paper, we presented a feature-based classification approach to select best and the right tools. In brief some key factors in this classification reply to these considerable questions:

1. Flexibility: Can the system adapt to how your organization does business? 2. Ease of Use: Will your people be able to use the tool without a couple of hours training? 3. Category: Into which classification of agile project management tools does it fit, and does that class match with the needs of your organization? 4. Responsiveness: How responsive is the organization? 5. Pricing: Does the pricing of the system match the value you will receive? 6. Features: Does the system have enough features to meet your current and future objectives [11]?

Then we classified them in a table based on comprehensive factors:

- Feature-driven Development: Some companies attempt to use a traditional tool that causes their project to be more complicated due to these tools don't support basic agile practices [12].

- Lifecycle Management: Storing project information in different multiple tools causes inaccurate results and prevents to comfortable real-time visibility [13].

- Cross-Functionality: It means to manage the requirements of the customers, programmers, testers, product owners, and other stakeholders in an integrated environment to enhance collaboration and consistency [14].

- Configuration with Flexibility: An agile management tool should let companies to organize, and plan according to their requirements.

- Simplicity: Like agile software project development, the simple one with ease of use is better, but the level of maturity is considerable.

- Enterprise Scale: In order to deployment of an enterprise, agile tools should be able to handle the project structure, tasks, defects and tests [15].

Finally, we classified results in a table and presented a model to select right agile tool based on features of agile software development tool and enterprise needs [17, 18, 19]. In this model, 3 key criteria is applied, 1- cloud ability which indirectly covers security, 2- Open source vs. proprietary, 3- Co-located teams vs. Distributed teams. Finally, at each leaf, a couple of agile tools which best fits in this situation is offered. Due to most of the tools even those which needs high security, have cloud and web based capabilities, our model is based on cloud.

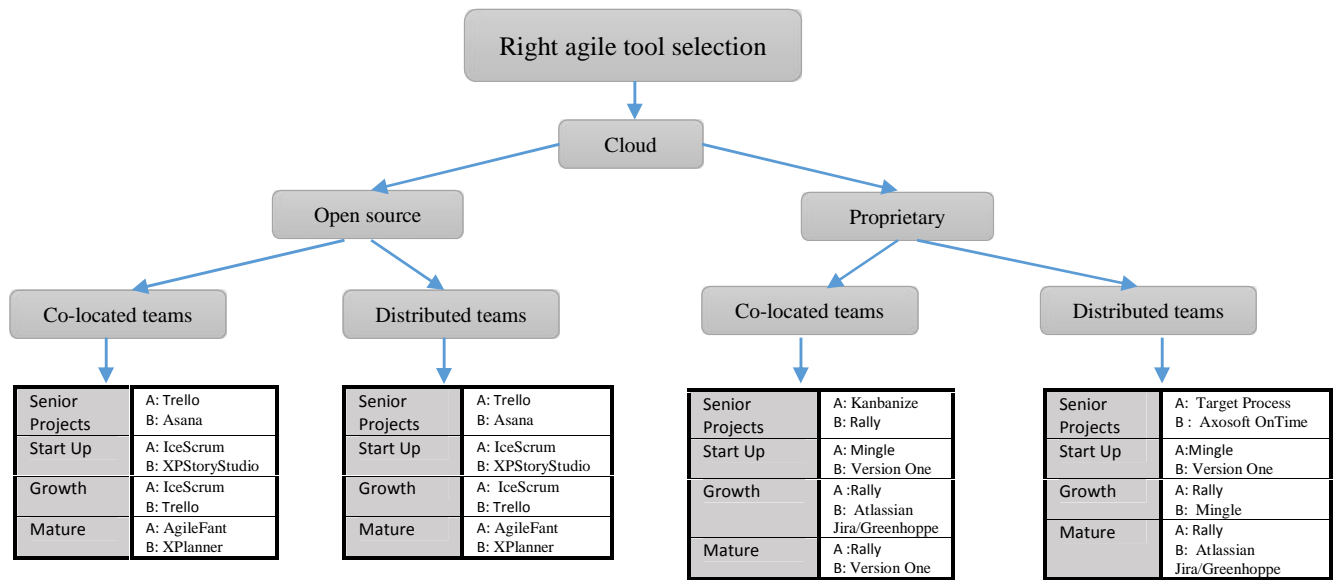


Figure 1. A selection model to choose the right agile tool on the cloud

ACKNOWLEDGMENT

This research was partly supported by the Leona M. and Harry B. Helmsley Charitable Trust via the Georgia Tech's Vertically Integrated Projects (VIP) Program. The authors also thank the FIU graduate students attended Dr. Sadjadi's Fall 2014 Advanced Software Engineering class for their technical assistance and insightful discussions during the class sessions. This material is based in part upon work supported by the National Science Foundation under Grant Numbers of I/UCRC IIP-1338922, AIR IIP-1237818, SBIR IIP-1330943, III-Large IIS-1213026, MRI CNS-1429345, MRI CNS-0821345, MRI CNS-1126619, CREST HRD-0833093, I/UCRC IIP-0829576, and MRI CNS-0959985.

REFERENCES

- [1] Martin, Robert Cecil. Agile software development: principles, patterns, and practices. Prentice Hall PTR, 2003.
- [2] Abrahamsson, Pekka. Agile Software Development Methods: Review and Analysis (VTT publications). 2002.
- [3] Azizyan, Gayane, Miganoush Katrin Magarian, and Mira Kajko-Matsson. "Survey of agile tool usage and needs." Agile Conference (AGILE), 2011. IEEE, 2011.
- [4] Azizyan, Gayane, Miganoush Magarian, and Mira Kajko-Mattsson. "The Dilemma of Tool Selection for Agile Project Management." ICSEA 2012, The Seventh International Conference on Software Engineering Advances. 2012.
- [5] Behrens, Peter. "Agile Project Management (APM) tooling survey results." Trail Ridge consulting (2006).
- [6] Dubakov, Michael, and Peter Stevens. "Agile Tools: The Good, the Bad and the Ugly." Report, TargetProcess, Inc (2008).
- [7] VersionOne.com. "The 8th Annual "State of Agile" Survey."

- [8] <https://www.youtube.com/channel/UCucIzz2RRY6tYcKuTDxO8pw>
- [9] Kumar, Misha, Laszlo Huber, and Milan M. Jovanovic. "Start-up procedure for three-phase six-switch boost PFC rectifier." Applied Power Electronics Conference and Exposition (APEC), 2014 Twenty-Ninth Annual IEEE. IEEE, 2014.
- [10] Giardino, Carmine, et al. "What do we know about software development in startups?." Software, IEEE 31.5 (2014): 28-32.
- [11] Bustard, David, George Wilkie, and Des Greer. "The maturation of agile software development principles and practice: Observations on successive industrial studies in 2010 and 2012." Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the. IEEE, 2013.
- [12] Thakur, Siddharth, and Harshavardhan Singh. "FDRD: Feature driven reuse development process model." Advanced Communication Control and Computing Technologies (ICACCT), 2014 International Conference on. IEEE, 2014.
- [13] Reichert, Manfred, Alena Hallerbach, and Thomas Bauer. "Lifecycle management of business process variants." Handbook on Business Process Management 1. Springer Berlin Heidelberg, 2015. 251-278.
- [14] K Majchrzak, Ann, Philip HB More, and Samer Faraj. "Transcending knowledge differences in cross-functional teams." Organization Science 23.4 (2012): 951-970.
- [15] Ambler, Scott W., and Mark Lines. Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise. IBM Press, 2012.
- [16] Misha, Laszlo Huber, and Milan M. Jovanovic. "Start-up procedure for three-phase six-switch boost PFC rectifier." Applied Power Electronics Conference and Exposition (APEC), 2014 Twenty-Ninth Annual IEEE. IEEE, 2014.
- [17] Abrahamsson, Pekka, Nilay Oza, and Mikko T. Siponen. "Agile Software Development Methods: A Comparative Review1." Agile Software Development. Springer Berlin Heidelberg, 2010. 31-59.
- [18] Langer, Tomáš, and Pavel Vaněk. "AGILE METHODS IN TECH-STARTUP." IMEA 2012
- [19] Turk, Dan, Robert France, and Bernhard Rumpe. "Limitations of agile software processes." arXiv preprint arXiv:1409.6600 (2014).

Adoption of Software Product Line to a Voice User Interface Environment

Diógenes R. F. Oliveira, Byron L. D. Bezerra, Elyda L. S. X. Freitas, Alexandre M. A. Maciel
Polytechnic School of Pernambuco – University of Pernambuco
Recife, Brazil
{drfo, byronleite, amam}@ecomp.poli.br, elyda.freitas@upe.br

Abstract — Software Product Line is a software development paradigm created to meet different market segments. This paradigm has shown great acceptance in the corporate environment (Motorola, Nokia, and Hewlett Packard) to allow the construction of more efficiently through reusing common components applications, besides being extensively researched by academics. The segment of voice interface, in turn, came up with the demand for systems capable of interacting with users, but in the application development process for this domain there is a lack of tools that make the task more productively. The FIVE (Framework for an Integrated Voice Environment) is a development environment for Voice Interface products designed to increase productivity in this segment. This paper aims to apply a SPL approach to FIVE. For this, a comparative evaluation of the process of construction of FIVE and SPL platforms was performed. Then adjustments in order to correct structural problems and, finally, the framework was validated using a set of experiments which sought to ensure the confirmation of such changes have been made.

Keywords: *Experience Report,s Software Product Line; Voice User Interface.*

I. INTRODUCTION

In recent years, the area of Voice User Interface (VUI) has received great attention from academics, for two main reasons: first, due to improvements in the performance of automatic speech processing systems, including speech recognition and speech synthesis; secondly, due to convergence device and mass production of multimedia content, which requires means of user interaction faster and efficiency [1].

According to Huang *et al.* [2], the typical architecture for the development of VUI has three components: the first represents the set of engines responsible for the speech recognition or the speech synthesis; the second consists of a API (Application Programming Interface) used to facilitate communication between engines and applications; and the last one consists of a set of possible applications. This architecture has guided this area over the years and many resources have been created with the aim to assist in this process.

Much has been done, both academia and in industry to provide improvements in speech recognition rates and speech synthesis naturalness, however, little effort has been made to bring these advances at the application level. Given this scenario, was developed the FIVE (Framework for an Integrated Voice Environment) in order to assist in speech

engines building and in instantiation of them in different technological environments (Telephone, Mobile, SmartTV) [3].

The FIVE has been used by the company Vocal Lab in a real development environment. With him, the time-to-market was considerably reduces and enable the mass development of products with voice interface. The Voc Refactoring the Environment al Lab, offers a products family for speech recognition (*VL Recognizer*), speech synthesis (*VL Synthesizer*) and speaker verification (*VL Identifier*).

According to Pohl [5] Software Product Line (SPL) is a set of software systems that have a certain set of features in common, and meet the needs of a particular market segment or mission and are developed with the same core assets. Although it is not explicit in the original work of Maciel [4], FIVE presents a SPL behavior, however, various features of SPL presents some problems inherent in this approach, as:

- Lack of variability management, which causes a lack of control of altered or removed features;
- Severe failure of the features configuration, generating products with errors and / or locking tool.
- No identification of features, preventing management for maintenance and evolution.

Given these problems, FIVE does not function properly as SPL in all family products. This leads to lost productivity, reducing the potential of time-to-market as suggested by the tool. In this sense, this paper aims to propose the adoption of Software Product Line approach in FIVE. For this, this paper is organized as follows: Section 2 provides a background to the literature of SPL adoption. Section 3 shows adoption process. Section 4 shows the experiments performed for the FIVE and his adaptation to the concepts of SPL and finally section 5 describes conclusions.

II. SOFTWARE PRODUCT LINE ADOPTION

The adoption the SPL concept emerged together with the practice of software reuse. In 1983, Doe and Bersoff [6] presented the software industry an initiative to increase productivity and quality by creating an environment composed of techniques and tools to assist the process of software development with reuse. The literature on the adoption of software product line is enough extensive. Bosch [7] reports the adoption of alternatives is generally much more diverse than those presented in the literature and the technical and

organizational criteria for adoption have more freedom than we might expect.

This diversity can be seen through an analysis of the major works published in recent decades. Bosch in [7] created a maturity model served as a reference in the evolution of product lines. Clements and Northrop [8] synthesized the fundamentals of SPL, practices and standards used, which provided a model with the essential approach to application of SPL. Linden et al. [9] presents the best practices of the industry in the adoption of SPL using the foundations created by Clements and Northrop, which provide practical actions used SPL processes. Finally, more recently, Apel *et al.* [10] present a features-oriented model for SPL with concepts and practical implementations.

Despite the freedom in SPL adoption, these models have common areas: *Domain Engineering*, responsible for collecting, organizing and storing past experience in systems development activities; and the *Application Engineering* responsible for the process where applications are built by reusing the artifacts of Domain Engineering and by exploration the variability [5]. Given this scenario, this work has been inspired Pohl *et al.* [5] and Apel *et al.* [10] to making the decisions about the necessary actions that would be applied to FIVE environment.

III. ADOPTION PROCESS

Considering the reading of the Hall of Fame of SPL adoption we propose a adoption methodology composed of four steps: Interview with Experts; Evaluation by Inspection; and Refactoring the Environment.

A. Interview with Experts

The purpose of these interviews was to obtain qualitative information regarding conceptual and architectural issues of FIVE as a SPL solution. Three researchers were selected with practical proven of the SPL approach, and by the vast theoretical knowledge through the publication of articles, consultancies and projects.

At this stage it was possible to better understand existing problems in the framework and know the possible actions that could be taken to solve the problem of traceability of artifacts of the tool. Two key points were mentioned by specialists. First one was the need to identify the features that FIVE pretend to present within the domain of VUI. The second point raised was the need to implement a Configuration Knowledge in order to manage the features and the dependencies control. According to experts, these actions guarantee the stable operation of FIVE by defining constraints as the variability of the components of the platform should compose the derivation of products.

B. Evaluation by Inspection

Pohl [5] propose two steps for SPL adoption: evaluation of Domain Engineering and Evaluation of Application Engineering.

1) Evaluation of Domain Engineering

a) Product Management

At this step, we analyzed the original work of Maciel [4] and reports on the design of FIVE had scoped the area of VUI.

Tool's market strategy would provide developers, a productive mechanism, with fast learning curve and multi-platform. This observation was confirmed from the many difficulties on the part of developers, which apply in systems the interface models. A more detailed evaluation of the product was carried out, using three basis listed by Pohl [5] to be observed in the phase of product management activities. Table 1 shows the result of the evaluation.

TABLE I. EVALUATION OF PRODUCT MANAGEMENT

Activity	Description	Application in FIVE
Scope of product portfolio	Determines the range of products to be offered	Engines for speech recognition, speaker verification and speech synthesis.
Scope domain	Identifies major functional areas that are relevant to SPL	Pattern Acquisition, Feature Extraction, Pattern Classification and Engine Generation.
Scope core asset	Define the required features of components	Functionalities needed are techniques for feature extraction and pattern classification.

b) Requirements Engineering Domain

In FIVE, requirements engineering domain were made through the use of questionnaires, interviews and surveys with VUI experts. Thus, the notation used for specifying the specific language was the domain of VUI. Based on the identified functional areas, process was a questionnaire made available in order to meet the difficulties, the needs, the environment, suggestions, among other aspects of the process of construction and application of models of voice interface. Table 2 shows the main requirements raised.

TABLE II. EVALUATION OF ENGINEERING DOMAIN

Scope Domain	Requirements
Pattern Acquisition	Features integrated audio recording and edition.
Feature Extraction and Pattern Classification	Provide mechanisms for analysis and comparison of techniques.
Engine Generation	Platform independence and ease of integration with applications.

c) Design Domain

The general architecture of FIVE was designed independently based on three modules: CORE consisting of a framework of classes where the central implementation of the framework; API that is a proprietary implementation that provides a set of resources needed to mediate between the speech engines and the application layer; and the GUI (Graphical User Interface) which is a graphical interface that assists the development of projects.

Although the proposed architecture meets in an adequate manner the generation of products, a failed architecture was identified. In Engine Generation step all features are loaded to the end product, with no differentiation in architecture on unused features.

d) Realization of the Domain

Previous study evaluated the implementation of FIVE that the best approach to implementing a mechanism for mass

production voice interface, according to know requirements would be through the mechanism Framework Gray-box [11].

The class diagram is centered on the Project Class that relates to the classes: Utterance, Sample, Speaker, Extraction, Classification and ProjectType. All these classes have their methods of adding, updating, deleting and research, except ProjectType which is just a class of type Enum. The Utterance class has a special behavior to receive the grapheme-phoneme of the utterance of helper classes Phrase, Word and Syllable are responsible for representing the linguistic details of each phrase. They make common variability in code level serving different products and product-specific aspects are addressed from use of the inheritance mechanism.

2) *Evaluation of Application Engineering*

The environment for generation of products FIVE is a platform that has the format wizard that has sequential tabs, the classic process of recognizing patterns defined by Duda et al. were inspired. [12]. The user-friendly interface, how they are arranged the information, the ease of applying the techniques of feature extraction and classification, and the definition of parameters, all these criteria together, decrease the learning curve in the generation of models for VUI applications.

For this evaluation were built several engines and it was observed that even with the use of the framework mechanism for implementing the variability of the components, it hurts not enough for a proper functioning of FIVE as a SPL. Thus, taking into account the background regarding the adoption of SPL and interviews with experts, if make know that other activities should be performed: the construction of the Feature Model and a Knowledge Configuration.

In the Feature Model to identify the features available, its constraints and dependencies occurs, and from it is possible to know the potential variability of the platform. Configuration Knowledge has the role of expressing the relationships and dependencies between the variables of the product line and features, and their interactions. Thus, observed the need of carrying a refactoring of FIVE environment given that, in its initial implementation, the proposed fast generate products was achieved, but there is a lack of the integrity of the products and the operation of the line from of features selected.

C. *Refactoring the Environment*

The process of FIVE refactoring the environment in two steps: first the Feature Model was developed and then implemented a mechanism of Configuration Knowledge.

1) *Development of the Feature Model*

The development of the Feature Model was accomplished with the aid of pure::variants tool [13]. The choice of this tool was made because it is widely used in both academia and industry. From the evaluation of all features FIVE and its dependencies were identified. Some numerical features were created in order to empirically parameter settings are adopted in the area of voice interface.

The Feature Model, which is attributed as the main element of the project itself FIVE contains five features as direct daughters, three of which were considered more complex due to the number of variations, specifically Rating Standards and

Feature Extraction, where various techniques are implemented and the internal variations caused mainly by setting parameters. Features like numbers are justified because of being parameters adopted in the literature for specific techniques.

2) *Implementation of Configuration Knowledge*

The development of the Knowledge Configuration happened according to the proposal of Domain-Specific Modeling of Cyril [14]. According to him the use of domain specific abstractions tends to facilitate the understanding of the variability, this mechanism has been used implicitly in FIVE, even to the point of an individual with experience in VUI applications do not need to run the platform.

According to the results of the evaluation of five to failure dependence between the features is the main problem in the platform. For example, given an "A" feature selected at a time, necessarily requires a "B" functionality later for maintenance operation. Thus, the implementation of a knowledge configuration meets solve this dependency failure.

The development of the Knowledge Configuration was done through crosstree constraints following three phases: change in GUI, register control and inclusion of features extracted from the field and ranked. In the original version of FIVE, the graphical interface allowed the indiscriminate selection of techniques for extracting's characteristic, regardless of the technique of pattern classification. To resolve this problem with a new interface control features available for selection was developed.

The control record of the features was necessary because the original version of FIVE, Each new selection of features for feature extraction, the data were overwritten, not allowing to have a history of previous features. To mitigate this problem adaptation was performed for selected features were stored in the corresponding techniques of extraction of features selected subdirectories.

The inclusion of the field extracted and classified was necessary as the original version of FIVE only the last feature extraction of selected features could be used for classification. To mitigate this problem it was tailored to data structure for addition of a new Boolean attribute (extracted). This solution allowed the use of any features extraction that are available.

IV. EXPERIMENTATION WITH DEVELOPERS

In this study, an observational assessment of the use of FIVE original and the new version after refactoring, followed by the application of a questionnaire was carried out. The purpose of this evaluation was to assess the implementation of FIVE experiencing the potential variability of products, specifically variants of techniques used in generating speech engines, both feature extraction as the classification of patterns.

The experiment with the collaboration of five developers who had no prior knowledge about the FIVE, however, all were familiar with the process of pattern recognition. The developers used the FIVE in the environment composed by Windows 8 operating system with as NetBeans with Java 7 Update 45 operating system, with all the default settings. FIVE were present in all the features of the production line required to build a product.

Observational assessment began with the realization with an orientation about the concept of SPL and it is the FIVE within the context of VUI, to equalize the knowledge of users. Then were distributed both versions of FIVE and requested the construction of a speech recognition engine for isolated words. Then a database of audio and text with five control commands (Open, Close, Follow, Stop, No) was available. The developers were free to choose the features for feature extraction and pattern classification. At the end, everyone was able to successfully generate the engines in both versions.

During the FIVE observational assessment metrics were used: time (in minutes), number of turns to earlier stages, the tool crashes, errors and doubts. Tables 3 and 4 present the results observed in two scenarios: evaluation with the original version, and evaluation with new version, respectively.

TABLE III. EVALUATION WITH THE ORIGINAL VERSION

Developer	Time	Turn Back	Crashes	Errors	Doubts
A	34	7	6	3	9
B	44	6	8	3	9
C	38	8	6	1	10
D	42	9	7	2	11
E	43	10	6	3	9

TABLE IV. EVALUATION WITH NEW VERSION

Developer	Time	Turn Back	Crashes	Errors	Doubts
A	26	7	0	0	6
B	35	7	0	0	8
C	31	5	0	1	6
D	32	7	1	0	7
E	28	9	0	0	6

It is observed that the average for the construction of speech in scenario 2 engine time was 25% faster than in scenario 1, Although, the generation of speech remained in a short period of time engines. The number of turns was similar in both scenarios. The crashes were practically used, since their occurrence occurred due to the absence of Configuration Knowledge, specifically in the areas of feature extraction and pattern classification. The crash that occurred with the user D in scenario 2 was due to internal problems with the operating system. Errors in scenario 2 were reduced because the account Configuration Knowledge and doubts about the features decreased smoothly in scenario 2 since at that time there was already a greater familiarity with the tool.

V. CONCLUSIONS

A major strength of this study was the exploration of the SPL approach in the field of VUI, since this area is little explored by Software Engineering. Another important contribution was the refactoring of FIVE to make it really a SPL. With this the FIVE passes to carry around a set of values, among them, the possibility of development of research techniques of extraction and classification, because reading from the perspective of oriented features.

The correction of faults in the functioning of FIVE process, through the Knowledge Configuration and Feature Model solved the problems found in the previous version by defining the constraints of the features. The identification of features and construction of the model feature that provides visualization and potential of the platform.

In the experiments the features were willing to users only in accordance with the availability of the same features as the previously chosen, proving the importance of Configuration Knowledge for the correct operation of the platform and product generation correctly. After the restructuring, the FIVE happened to have a clear definition as to its engineering software, making their understanding for researchers and developers easier.

VI. REFERENCES

- [1] Kurniawati, E.; Celetto, L.; Capovilla, N.; George, S., "Personalized voice command systems in multimodal user interface," Emerging Signal Processing Applications (ESPA), 2012 IEEE International Conference on , vol., no., pp.45,47, 12-14 Jan. 2012
- [2] Huang, X., Acero, A., Hon, H.W., Spoken Language Processing – A Guide to Theory, Algorithm, and System Development, Prentice Hall, 2001.
- [3] Maciel, A.; Carvalho, E.; FIVE – Framework for an Integrated Voice Environment, IWSSIP, 2010.
- [4] Maciel, A., Investigação de um ambiente para o desenvolvimento integrado de interface de voz. Tese de doutorado, CIn/UFPE, 2012.
- [5] Pohl, K.; Böckle, G.; Van Der Linden, F.: Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Heidelberg 2005.
- [6] Doe, D. D. and Bersoff, E. H. (1986). The Software Productivity Consortium (SPC): An industry initiative to improve the productivity and quality of mission-critical software. Journal of Systems and Software, 6(4), 367–378.
- [7] Bosch, Jan. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization. Software Product Lines, 257-271, 2002, Springer Berlin Heidelberg
- [8] Northrop, L. M. e Clements, P.C. A Framework for Software Product Line Practice. Version 5.0. Pittsburg. Software Engineering Institute, 2007. Disponível em: <http://www.sei.cmu.edu/productlines/framework.html >. Acesso em: 02/12/ 2013 às 10:24.
- [9] Linden, Van der; Frank J., Schmid, Klaus; Rommes, Eelco. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer-Verlag Berlin Heidelberg, pp. 26, 2007.
- [10] Apel, S., Batory, D., Kstner, C., and Saake, C. Feature-Oriented Software Product Lines: Concepts and Implementation. Springer Publishing Company, Incorporated. 2013.
- [11] Fayad, M. E.; Schmidt, D. C.; Johnson, R. E. Building Application Frameworks: Object-Oriented Foundations of Frameworks Design. New Jersey: Wiley, 1999.
- [12] Duda, R.O., Hart, P.E., Stork, D.G., Pattern Classification, Wiley-Interscience. 2000.
- [13] pure::variants, available em: <http://www.pure-systems.com/> Acessado em Março de 2014.
- [14] Cirilo E., Nunes, I.; Kulesza, U.; Lucena, C.; Automating the product derivation process of multi-agent systems product lines. No SBES '09, página 12, Brasil, 2009. IEEE.

Adopting Agile Methods in the Public Sector: A Systematic Literature Review

Isaque Vacari

Embrapa Informática Agropecuária
Empresa Brasileira de Pesquisa Agropecuária, Embrapa
Campinas/SP, Brazil
isaque.vacari@embrapa.br

Rafael Prikladnicki

Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre/RS, Brazil
rafaelp@pucrs.br

Abstract - Agile software development (ASD) has become an important research topic. However, despite the increase in the number of studies in this area in the last few years, there is a lack of structured information about its adoption in the public sector. Since the public sector is the part of the economy concerned with providing various government services, the goal of this study is to report from a systematic literature review and provide information that may enhance the understanding of the implications of adopting ASD within public companies. As the main results, we found that ASD could indeed be adopted in the public sector. The analysis suggests that a good alternative is to start the adoption of ASD with people willing to change - strongly supported by senior management - working on important pilot-projects. Second, we found that job satisfaction is greater when adopting agile methods within public companies. Finally, we also found some barriers that are difficult to overcome, including the ingrained use of plan-drive methods, as well as big bang deliveries and lack of experience in ASD.

Keywords: agile methodologies; software development; software engineering; public organizations; government.

I. INTRODUCTION

The public sector is the part of the economy concerned with providing various government services. With the development and evolution of technology in the past few years, public organizations began to gradually incorporate software products in their development processes. Thus, the incentive to adopt new and better approaches to Software Engineering (SE) has become something essential for the future of software projects in government. In the context of SE, the government always attempted to enforce software standards for the development of its systems, based on plan-driven methods with big bang deliveries. For example, the United States Department of Defense attempted to enforce "US Military Software Development Standards" from the 1970s to 1990s [1]. Also, the United Kingdom (UK) Government decided to adopt "Structured Systems Analysis and Design Method" from the 1980s to mid-2000s [2]. The benefits are that it contributed to spreading the use of valuable techniques. On the other hand, software standards based on plan-driven methods are unlikely to cope well with uncertainty, changing requirements, user communication and staff development [2].

A good solution to these problems was formalized in February of 2001 with the agile methods. Since then the government has gradually waived its ways of working and has adopted agile software development [3]. Although ASD has

become an important research topic and the number of studies in this area has increased in the last few years, there is a lack of structured information about its adoption in the public sector. For this reason, the goal of this study is to report from a systematic literature review about the adoption of agile methods in the public sector and provide empirical evidence about its current situation, challenges, and opportunities.

Public organizations have been adopting agile methods in order to improve the results of their IT projects [7][8][9]. This is motivated by the benefits that agile methods can bring to organizations, including the ability to manage changing priorities, better alignment between IT and business objectives, enhanced software quality and increased customers / users / stakeholders satisfaction with the software product [10].

However, the adoption of agile methods in the public sector has some challenges. Agile methods are incompatible with hierarchical and bureaucratic structures, typical of government [11]. Many public organizations, especially large ones, have spent years changing their culture so that the processes were defined and followed, it is difficult to switch to a working model in which the processes are informal and defined by the development team [3]. Even so, agile methods can bring better results for the public sector, than those that would be possible to achieve with plan-drive methods big bang deliveries [3].

This paper organized as follows: in Section II, we present the method. In Section III, we set out the results, while in Section IV we discuss the findings. In Section V, the limitations of the study are discussed. Finally, in Section VI the conclusions and future work are addressed.

II. RESEARCH METHODOLOGY

The research methodology used is a systematic literature review. The main purpose was to find evidence regarding the adoption of agile methods in the public sector. Our review protocol was based on the recommendations provided by Kitchenham [4], and the research question that guided the systematic review was:

What is known about the adoption of agile methods in the public sector?

The search included digital libraries available and papers published in journals, conference and workshop proceedings. We searched seven digital libraries and one-conference proceedings, the following: ACM Digital Library, Bielefeld

Academic Search Engine (BASE), ScienceDirect, Engineering Village, IEEEExplore, Scopus, SpringerLink, Web of Knowledge and Wiley. In extra, we hand-searched Portuguese studies for research papers: *Bases de Dados da Pesquisa Agropecuária (BDPA)*, *Biblioteca Digital Brasileira de Computação*, *Workshop Brasileiro de Métodos Ágeis (WBMA)*.

The keywords were defined based on two main categories of terms: those related to “Public Sector”, and those related to “Software Development”. Table I outlines the keywords.

TABLE I. KEYWORDS USED IN THE REVIEW PROCESS.

Reference	Category	Keywords
A	Public Sector	Government (1) Public sector (2) Public administration (3) Public organization (4)
B	Software Development	Software development life cycle (5) Software development methodology (6) Software development process (7) Software development projects (8) Software process (9) Unified process (10) Rational unified process (11) RUP (12) Microsoft solutions framework (13) Agile methodologies (14) Agile methods (15) Agile principles (16) Agile process (17) Agile software development (18) Extreme programming (19) Lean software development (20)

The search was a combination of A and B. Category B has more keywords and reflects the fact that there are many variations of the same term. We defined the search string as:

$(1 \text{ OR } 2 \text{ OR } 3 \text{ OR } 4) \text{ AND } (5 \text{ OR } 6 \text{ OR } 7 \text{ OR } 8 \text{ OR} \dots 20)$

Figure 1 shows the systematic review process and the number of papers identified at each stage.

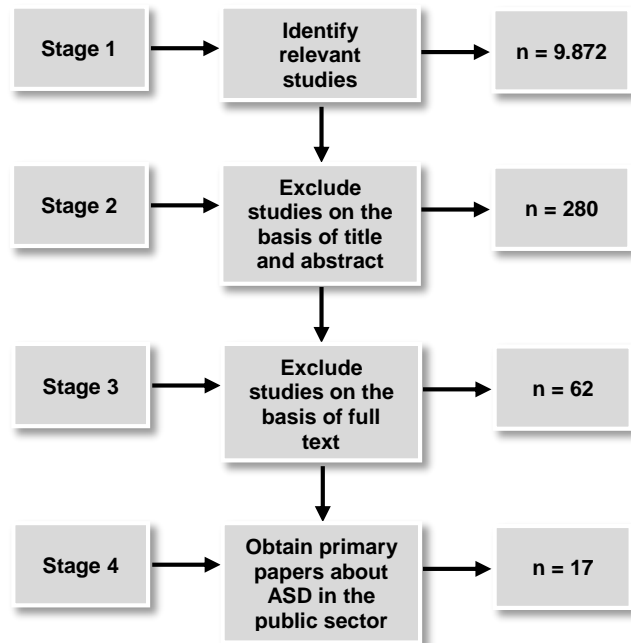


Figure 1. Stages of the study selection process.

We searched for government experience reports and empirical papers. This search strategy resulted in a total of 9.872 citations. As we can see, the lack of standard terminology in “Public Sector” and “Software Development” resulted in a large number of papers to start with, but only a few were selected, confirming the high sensitivity and low precision of our search.

To include a paper in the analysis, the paper must have been available online, must have been written in English or Portuguese, and must have described the aspects of the adoption of agile methods in the public sector. The papers were classified following a three-step approach. First, based on the reading of the papers’ titles and abstracts, the papers were classified into three categories:

- [Inc], indicating the papers collected and possibly related to software development in the public sector.
- [Exc], indicating the papers collected but not related to software development in the public sector.
- [Dup], indicating the papers collected but repeated with other studies.

All the papers in category [Exc] or [Dup] were excluded, while the papers in category [Inc] were analyzed more carefully based on the reading of the full text (introduction, conclusion, and specific parts related to the main contribution). Then, a subset of papers in [Inc] related to software development in the public sector was selected for the next step. Finally, a new subset of papers in [Inc] was selected, keeping only those addressing regarding agile software developments in the public sector.

After this process, the papers were classified according to three general categories of information:

- **General information:** digital library, title, authors, source (e.g. journal or conference proceedings) type of source (i.e. journal, conference, workshop, technical report), and category ([Inc] or [Exc] or [Dup]).
- **Research-related information:** type of paper (i.e. theoretical, industrial experience report, or empirical study), research empirical strategy (i.e. case study, survey, experiment, ethnography, action research, combination), data collection methods (i.e. interview, observation, questionnaire, document inspection, or multiple data collection methods), type of data analysis (i.e. qualitative, quantitative or both), and data analysis method (i.e. statistics, grounded theory, content analysis). For papers reporting empirical work, the type of study was classified according to the proposal in Dias Neto et al. [5]. Research strategy, data collection, type and method of data analysis were classified according to the terminology used by Oates [6].
- **Content-related information:** aspects of the adoption of agile methods (i.e. reasons, benefits, problems and challenges, recommendations), project features (i.e. team size, agile method, agile experience, project duration, domain, customers, contractors), attributes, and general comments.

TABLE II. PAPERS SELECTED FOR ANALYSIS.

<i>Study</i>	<i>DL</i>	<i>Title</i>	<i>Authors</i>	<i>Year</i>
[P01]	Scopus	A case study: Introducing eXtreme programming in a US government system development project	A. Fruhling, P. McDonald, and C. Dunbar	2008
[P02]	Scopus	Staying agile in government software projects	B. Upender	2005
[P03]	WBMA	Adoção de métodos ágeis em uma Instituição Pública de grande porte - um estudo de caso	C. de O. Melo, and G. R. M. Ferreira	2010
[P04]	Scopus	The FBI gets agile	C. Fulgham; J. Johnson; M. Crandall; L. Jackson; N. Burrows	2011
[P05]	Scopus	Collaborative development of public information systems: A case study of "Sambruk" e-services development	C.-O. Olsson, and A. Öhrwall Rönnbäck	2010
[P06]	IEEE	Making agile development work in a government contracting environment-measuring velocity with earned value	G.B. Alleman, and M. Henderson	2003
[P07]	Scopus	Agile development in a bureaucratic arena - A case study experience	H. Berger	2007
[P08]	Scopus	An industrial case study for Scrum adoption	H. Hajjdiab, A. S. Taleb, and J. Ali	2012
[P09]	Scopus	Army simulation program balances agile and traditional methods with success	J. Surdu, and D.J. Parsons	2006
[P10]	BASE	A Case Study on the Adoption of Measurable Agile Software Development Process	M. Iliev, I. Krasteva, and S. Ilieva	2009
[P11]	BDPA	Extreme Programming by example	M. Pedroso Jr, M. C. Visoli, and J. F. G. Antunes	2002
[P12]	Scopus	Lessons learned using agile methods on large defense contracts	P. E. McMahon	2006
[P13]	IEEE	Evolving to a "lighter" software process: a case study	R. J. Moore	2001
[P14]	Springer	Is Agile the Answer? The Case of UK Universal Credit	R. Michaelson	2013
[P15]	Scopus	Agile software development under university-government cooperation	S. Kaneda	2006
[P16]	Scopus	Exploring XP for scientific research	W. A. Wood, and W. L. Kleb	2003
[P17]	Scopus	Agile metrics at the Israeli Air Force	Y. Dubinsky, D. Talby, O. Hazzan, and A. Keren	2005

TABLE III. OVERVIEW OF THE STUDIES.

<i>Study</i>	<i>Research Method</i>	<i>Agile Method</i>	<i>Agile Experience</i>	<i>Project duration</i>	<i>Team size</i>	<i>Co-located</i>	<i>Contractors</i>	<i>Domain, comment</i>
[P01]	Case study	XP	Beginner	-	5	Yes	Yes	Events management
[P02]	Case study	Scrum and XP	Beginner	24 months	7	-	Yes	Health
[P03]	Case study	Scrum and XP	Beginner	14/10 months	7/6	Yes/Yes	No/No	Bank, financial management
[P04]	Case study	Scrum	Beginner	72 months	45	Yes	Yes	Cases management
[P05]	Multicase	Scrum	-	-	-	No	No	Help desk
[P06]	Case study	XP	-	-	-	-	-	-
[P07]	Mixed	-	Mature	36+ months	50+	Yes	-	IT mega project
[P08]	Case study	Scrum	Beginner	NA	NA	Yes	No	Adopting agile methods
[P09]	Case study	-	Mature	-	26 teams	Yes	Yes	Modeling and Simulation
[P10]	Case study	-	Beginner	-	16+	No	No	Distributed system
[P11]	Case study	XP	Beginner	-	-	No	Yes	Projects management
[P12]	-	-	Mature	NA	NA	NA	NA	Large defense contracts
[P13]	Case study	XP	Beginner	-	-	-	No	-
[P14]	Case study	-	Beginner	18 months	-	-	-	Payment of social benefits
[P15]	Case study	-	Beginner	-	-	-	-	Events management, Web-GIS
[P16]	Experiment	XP	Beginner	-	2	Yes	No	Aerospace engineering
[P17]	Case study	XP	Beginner	12 months	60	-	-	-

III. RESULTS

The systematic literature review was conducted from August 2013 to March 2014 and we identified 17 primary studies on agile software development in the public sector (see Table II). Key data, along with a description of the domain in which each study was conducted, is presented in Table III.

We categorized the studies into three main groups: (1) reason and benefits obtained from adopting agile methods in the public sector, (2) problems and challenges faced from adopting agile methods in the public sector, and (3) lessons learned obtained from adopting agile methods in the public sector.

A. Reasons and benefits obtained from adopting agile methods in the public sector

One of the biggest reasons for the adoption of agile methods is the benefits that they can bring to the government, which are a response to a history of failure of IT projects in public sector. Thirteen studies described some benefits obtained from adopting agile methods in the public sector. Table IV presents these studies.

TABLE IV. BENEFITS OBTAINED FROM ADOPTING AGILE METHODS IN THE PUBLIC SECTOR.

Factor	Study
Deliver value to customers/stakeholders earlier	[P02] [P09] [P12] [P13]
Better collaboration between IT and business	[P01] [P02] [P04] [P05] [P07] [P09] [P10] [P17]
Improved customer/stakeholder satisfaction	[P01] [P02] [P03] [P04] [P10]
Improved team morale and reduced dependence on contractors	[P03] [P11] [P17]
Improved communication	[P01] [P02] [P04] [P09]
Improvement in learning new technologies	[P03]
Improved product quality	[P16]
Improved project visibility	[P02] [P12] [P17]
Increased productivity	[P03] [P16]
Reduced cost	[P07]
Improved manage changing priorities	[P05] [P17]

Based on Table IV, we can observe that benefits were reported in the following areas: customer/stakeholder collaboration and alignment between IT and business object. Some studies have found that job satisfaction is greater, developers are more satisfied with their job and that customers are more satisfied with the product. However, a study did not find any benefit, and the evidences suggest that not all development environments have evolved with the same pace. For this reason, an organization's inherent culture may not match the development approach adopted, causing project failures [P07].

B. Problems and challenges faced from adopting agile methods in the public sector

In some cases, the optimistic view of agile methods can be imposed by a practical reality dominated by problems and challenges. Table V presents some of problems and challenges found.

TABLE V. PROBLEMS AND CHALLENGES FROM ADOPTING AGILE METHODS IN THE PUBLIC SECTOR.

Factor	Study
Organizational culture	[P01] [P02] [P03] [P07] [P08] [P14] [P15] [P16] [P17]
Lack of knowledge and experience with agile methods	[P01] [P02] [P03] [P08] [P11] [P14] [P17]
Little or no involvement of customers/stakeholders	[P07]
The ingrained use of prescriptive approaches and big bang deliveries	[P01] [P03] [P04] [P08] [P15] [P17]
IT mega projects	[P04] [P14]
Traditional procurement and contracts	[P01] [P03] [P04]
Compliance with standards and regulations	[P04] [P06] [P09] [P12]
The lack of senior management support	[P03] [P07] [P08] [P16] [P17]
Delays	[P01] [P04] [P07] [P08]

Based on Table V, the organizational culture is a primary factor for the success of any change initiative, including the adoption of a new approach to developing software. Although the culture incorporates many facets, we want to highlight just one specific part, since it presents relevant elements to agile methods in the public sector. This part is "The ingrained use of prescriptive approaches and big bang deliveries". Although various aspects of adaptive development have been advocated and valued by the US and UK Government lately [7] [8], there is still a bias towards prescriptive approaches and big bang deliveries in public sector. The roots of this trend are often associated with traditional procurement and contracts [3]. The US Government has to follow procurement processes that exacerbate the tendency to big projects, prescriptive approaches and big bang deliveries [3]. On the other hand, recently, the UK Government published a new clarification of business case guidance, explaining how government organizations get permission to spend money on agile work, supporting an agile culture in the public sector [9].

Moreover, some studies reported that agile software development practices are easy to understand, but applying them in practice can be difficult.

- Teams did not recognize value in Pair Programming
Studies: [P01] [P11]
- Teams encountered difficulties to write unit tests
Studies: [P03]
- Teams did not adopt Test-Driven Development (TDD) because of the lack of real examples
Studies: [P11]
- Teams have not found benefits on acceptance testing for web applications, due to frequent changes
Studies: [P02]
- Teams faced difficulties in convincing customers to deploy partial system in production, even if they add value to business
Studies: [P03]

- Teams reported difficulties in implementing iterative and incremental development
Studies: [P17]

C. *Lessons learned from adopting agile methods in the public sector*

One of the best ways to increase organizational memory is by conducting a lessons learned session. The storage and dissemination of lessons learned make the organization of workers reflect on experiences and use organizational memory as a starting point for present and future decisions. Some lessons learned are presented in the following dimensions:

1) *Team*

- Agile methods require knowledge and experience of the team
Studies: [P01]
- Training and coaching contributes to the formation of teams with less experience in agile methods
Studies: [P01] [P02] [P03] [P08] [P09] [P17]
- Agile encourages the formation of small and interdisciplinary teams, which reduces the complexity of communication and the time for decision-making
Studies: [P04]
- Agile can be adopted by geographically distributed software development teams
Studies: [P05]

2) *Customer relationship*

- Agile requires commitment and preparation, both for customers and developers to achieve their full potential
Studies: [P05] [P11]
- Agile opens the software development process to the customers through a direct communication with developers
Studies: [P01] [P04] [P07] [P09] [P10] [P11] [P17]
- Agile requires timely communication, accurate and complete among all members of the development team, customer and end-users
Studies: [P01]
- The best solutions are created when the customer is actively involved in the software development
Studies: [P07]

3) *Relationship with business*

- Incremental and regular delivery of software are important to demonstrate the emerging solution for customers/stakeholders, which increases trust between all project members
Studies: [P02]
- Progress meetings to communicate project status are tailored for specific audiences
Studies: [P01]

- Agile requires earned value management to maintain compliance with government regulations at all levels
Studies: [P04] [P06] [P09] [P12]

- Agile metrics are essential to make the decision-making more efficient and are important to promote project visibility sooner
Studies: [P02] [P12] [P17]

- Agile contracts requires negotiable scope
Studies: [P04]

4) *Processes and practices*

- Agile practices are simple to understand, however, to internalize them and follow them is strictly difficult
Studies: [P02] [P03] [P08] [P17]
- Agile achieves better results when government experts and developers work together with contractors in all phases of software development, in the same physical location and near customers
Studies: [P04] [P09]
- Pair Programming can be used in specific tasks of software development
Studies: [P02] [P16]
- To support Pair Programming the layout of the rooms of the developers needs to be changed
Studies: [P16]
- Agile requires more written tests by developers, which are executed automatically
Studies: [P03] [P04] [P09] [P10] [P11] [P13] [P16] [P17]
- Agile does not require less written documents and can be combined with prescriptive approaches
Studies: [P02] [P09]
- User experience aspects need to be considered from the earliest iterations
Studies: [P01]

5) *Software tools*

- Collaborative tools facilitate the adoption of agile methods
Studies: [P03] [P09] [P11] [P13]

IV. DISCUSSION

In this section we discuss our findings as follows.

A. *Agile methods can be adopted in the public sector*

We found that agile methods could be adopted in the public sector. In some studies, the results were better than those possible to achieve with prescriptive approaches. This is partly because customers/stakeholders are no longer just at the fringes of software development, but actively shaped and guided the evolution of the end software product or service. Secondly, there is a learning element inserted into each delivery cycle and a job satisfaction [P01][P02][P03][P04][P10][P11][P16][P17].

B. Adopting agile methods in the public sector can be even more challenging

The analysis suggests that agile software development in the public sector can be more challenging because people with little experience need to direct projects towards success with positive results in the short term, and sometimes they do not have the support organization necessary, and still depend tightly on external coaches [P08].

C. Adopting agile methods in the public sector can be slower and complex

The studies that address the adoption of agile methods suggests that a good alternative is to start the adoption of agile software development with people willing to change - strongly supported by senior management - working on important pilot-projects. After, the change will depend on their interaction with other teams, in order to reach the vast majority of the organization, which can be slower and complex [P01][P03][P11][P16]. One study showed that implementation of agile methods across the organization is recommended only when the negative results remain constant in time [P04]. In general, big bang adoption approach is not recommended, being more prone to failure [P08].

D. There is a need for more studies related to agile procurement

In government, the emphasis is on risk management through rigorous procurement processes, which generally includes a prescriptive fixed-price and fixed-requirement contract between client (government) and supplier (software industry) [3]. This approach is in disagreement with agile principles. As a result, those using agile development in the public sector are opting for resource augmentation (use of a supplier's staff on a time and materials basis) to run agile projects internally [P04] [P09]. This creates an opportunity for researchers to explore and understand the degree to which this approach is useful or appropriate to agile procurement context.

E. Aspects of the adoption of agile methods in the public sector should be empirically evaluated

Agile methods have not been sufficiently tested and exploited in the public sector. In addition, there is a restricted set of scientific evidence to extract conclusive results. However, there are promising results from adopting agile methods in public sector. Therefore, there is an opportunity for SE researchers to understand how these methodologies are adopted in practice and which effects they generate, including their advantages and disadvantages.

V. LIMITATIONS

Systematic literature review is a useful method. However, as any other method, there are some limitations. We address three of them. First, we see that the primary studies selected in our systematic review present significantly more evidence on success than failure cases. This may have limited or influenced the results.

Second, the full text of the articles was obtained through the libraries at the Brazilian Agricultural Research Corporation (Embrapa), *Pontifícia Universidade Católica do Rio Grande do Sul* (PUCRS) and University of Campinas (Unicamp). For this reason, some studies that could be related with the subject were not analyzed because they were not accessible.

Finally, the immediate definition of search strategy makes knowledge as something well defined and explicit. However, new knowledge was discovered while performing a systematic review, which could have resulted in a reformulation of the search strategy, along with a new execution of the whole process with the new discoveries. This means that an initial description accurate of reality makes it difficult to apply this method successfully, since the knowledge can also be volatile, tacit and diffuse.

VI. CONCLUSIONS

In this paper we report from a systematic literature review about adopting agile methods in the public sector. We found that agile methods could be adopted in the public sector. However, not all the implications of adopting agile methods in the public sector are widely known. For this reason, as next steps we plan to interview ASD teams from public organizations, aiming at expanding "what is known about the adoption of agile methods in the public sector", proposing a set of recommendations for adopting agile methods in this context.

REFERENCES

- [1] C. McDonald, "From art form to engineering discipline? A history of US military software development standards, 1974-1998," *IEEE Annals of the History of Computing*, vol. 32-4, pp. 32-47, December 2010.
- [2] P. Middleton, "Managing information system development in bureaucracies," *Information and Software Technology*, vol. 41-8, pp. 473-482, June 1999.
- [3] B. Wernham, *Agile project management for government*. London: Maitland and Strong, 2012.
- [4] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering". Technical Report, p. 57. Keele University and Durham University.
- [5] A. C. Dias Neto, R. Subramanyan, M. Vieira, G. H. Travassos, Characterization of model-based software testing approaches, Technical Report TR – ES 713/07, COPPE/UFRJ, 2007.
- [6] B. J. Oates, *Researching information systems and computing*. CA: Sage Publications, 2006.
- [7] US GAO, "Software development: Effective practices and federal challenges in applying agile methods," United States Government Accountability Office, 2012.
- [8] UK NAO, "Governance for agile delivery," National Audit Office, 2012.
- [9] HM Treasury, "Guidance: Agile digital and IT projects: clarification of business case guidance," HM Treasury.
- [10] VERSIONONE, "8th Annual state of agile development survey". VersionOne, 2013.
- [11] J. Iivari and N. Iivari, "Organizational culture and the deployment of agile methods: The competing values model view," in: *Agile Software Development - Current Research and Future Directions*, Springer Berlin Heidelberg, 2010, pp. 203-222.

Modeling Framework for Developing and Testing Network Security Techniques against DDoS Attacks

Konstantin Borisenko^{#1}, Ivan Kholod^{#2} and Andrey Shorov^{#3},

[#]Faculty of Computer Science and Technology,
Saint Petersburg Electrotechnical University (LETI),
Professor Popov str. 5, St.Petersburg, 197376, Russia

¹borisenkoforleti@mail.ru, ²iholod@mail.ru, ³ashxz@mail.ru

Abstract — In the paper we introduce a hybrid system for simulating DDoS attacks and computer network protection techniques. The developed system makes it possible to create various network topologies, perform experiments with DDoS attack simulation, develop new protection methods and test the existing ones. The suggested system not only allows us to design virtual networks, but also makes it possible to connect real network nodes for improving the accuracy of the experiments.

Massive DDoS attacks often affect websites of governments and government bodies of various countries, websites of leading IT-corporations. The world leaders in the area of information security consider DDoS detection and DDoS resistance as a primary task in their research and developments.

To study DDoS attacks and to develop new defense mechanisms researchers mostly use simulation methods. However, DDoS attack simulation often causes problems of the accuracy of the attack simulation on application level. Furthermore, depending on the software installed on a server this server can function in a different way. Also multi-level network construction in reality often requires dozens of time and resources

Thus, we suggest to integrate simulation and testbed methods. Using simulation, we create attacking network and connect it to the real server. A virtual network is very similar to a real one. In case of using DDoS attack traffic generators defense can be placed only on attacked server. Our approach allows to develop defense mechanisms, which can be placed anywhere in the attacking network. Furthermore protection mechanisms can be architecture-dependent, which is important for performing experiments in a way very close to a real network. An important advantage is the possibility of connecting real nodes to a virtual network, which will improve the accuracy of our experiments and allow us to test different settings and types of servers.

System development was performed using the discrete-event simulation system OMNeT++. The INET library was used for making network settings and packet switching. The ReaSE library was completed for creating topology settings. The system has special network interface, which allows to redirect traffic from the simulated network to real network and vice versa.

The verification of the system was successfully made. Comparison was made between system and networks constructed using Planetlab.

Authors have made series of experiments with different attack scenarios: SYN-Flooding attack, HTTP attack. Experiments were made without any defense methods and with filtering methods (Ingress, Egress). The network topology for conducting experiments consisted of 7 routers, 204 clients and 1 real server. The delays between network nodes are equal to 1 microsecond.

Now consider the analysis of scenario experiments for SYN Flooding using the Egress Filtering method and without using protection techniques. During SYN-Flooding attack, 20% of the total number of network clients (40 computers) participated in the attack. SYN cookies were switched off on the server for a successful SYN Flooding attack. At the beginning of the attack, at the 10th second, the number of server applications is increasing because more and more virtual clients are starting to take part in an attack. Till the 15th second the server is coping with the task of processing all requests and then the TCP stack is overflowed and the server is unable to process the increasing flow of applications. In the period from the 15th second till the 57th second the server provided no response to all arriving SYN-packets.

A series of experiments has been performed with the use of a filter with 2, 3, or 4 routers in a virtual network. At the same time the clients attacking the server continuously were located in the local network of a router that did not use Egress Filtering. With the increase of the number of filters the power of a server attack decreased.

The developed system can be used for studying DDoS-attacks and the protection techniques against them. Network administrators can quickly and precisely reproduce a network they are servicing, execute load testing, estimate server stableness to attacks, network capacity, and the quality of protection mechanism performance.

The paper has been prepared within the scope of the state project "Organization of scientific research" of the main part of the state plan of the Board of Education of Russia as well as project part of the state plan of the Board of Education of Russia (task # 2.136.2014/K).

Natural Language Processing to Quantify Security Effort in the Software Development Lifecycle

Constantine Aaron Cois
Carnegie Mellon University (CMU)
Software Engineering Institute (SEI)
Pittsburgh, PA, USA
cacois@andrew.cmu.edu

Rick Kazman
University of Hawaii and
SEI/CMU
Honolulu, HI, USA
kazman@hawaii.edu

Abstract—Addressing security in the software development lifecycle is an ever-present concern for software engineers and organizations. From a management and monitoring perspective, it is difficult to measure 1) the amount of effort being focused on security concerns during active development and 2) the success of security related design and development efforts. Such data is simply not recorded. If reliable measurements were available, software project leaders would have a powerful tool to assess risk and inform decision making. This would enable managers to direct development and testing to assure a desired level of security in their software products, to protect both their organizations and customers. To fill this need and provide such data, we propose a technique for performing topic detection on data commonly available in most software development projects: text artifacts from issue tracking and version control systems. We apply machine learning and natural language processing techniques to create classifiers capable of accurately detecting whether a given text snippet is related to the topic of security. Realization of such a capability will give software teams the ability to analyze current and past levels of security effort, revealing immediate project focus and the long-term impacts of security tasking. We validate our approach via experiments on data from the large-scale open source Chromium software project. Our results show that a Naïve Bayes classification scheme using an n-gram feature-space is an appropriate and effective approach to automated topic detection of software security text snippets, and that effective training data can be derived from public data sources without the need for manual intervention.

Keywords—*natural language processing; machine learning; software security; security; topic detection; classification; naïve bayes*

I. INTRODUCTION

Adequately injecting security into the software development lifecycle (SDLC) to ensure the creation of secure systems is a growing concern. Recent high profile security breaches in major industry and government organizations [1, 2, 3] have shown the unsettling vulnerability of modern software systems, even those developed by mature technology firms with experienced software engineering teams. Industry has taken note, and is responding with initiatives such as Microsoft's Security Development Lifecycle (SDL), a framework for formalizing and monitoring regular security effort throughout the SDLC [4]. In the open source community,

equally prolific security holes have recently been exposed, including the high profile Heartbleed [5, 6] and Shellshock [7] vulnerabilities, which contribute to a broad threat faced by many technology sectors and industries.

It is widely accepted that to achieve a high level of quality with regards to security in a software application, security as a quality attribute must be consistently addressed through all phases of the SDLC [4, 8, 9]. Techniques such as threat modeling, attack surface reduction, and penetration testing have been developed and put into practice in an attempt to meet this condition of secure software development [4, 10]. However, these techniques represent the injection of isolated activities at specific phases of the SDLC, not an overarching, consistent amount of concern for and effort towards security throughout software development. The goal of consistent security thought and effort in software development is hampered by the lack of robust means of identifying and measuring security effort within a software project or team.

Despite the efforts to regularize the reporting of security bugs (e.g., the CVE classification [11]), software development teams rarely record security bugs, tasks, or effort specifically. For this reason, accurate measurement of security effort is virtually impossible. In fact, our analysis of over 400,000 project repositories hosted on Github, a popular open source project management system, showed that only 1.4% of projects using a labeling system for tasks made available to developers a label for security related issues. The ability to identify and measure, at any point in time, the amount and type of effort being dedicated to security would give software developers and project managers powerful new capabilities. For example, they could plan and track the levels of effort expended towards software security throughout the SDLC. Additionally, such capability would allow teams to recognize early when projects are at risk of lowering security quality through inattention, thus avoiding the unintentional injection of vulnerabilities into their software product. Further, if applied continuously (or even in hindsight), such a quantitative capability would allow architects and project managers to identify points of introduction of design flaws, process failures, architectural inconsistencies, or weakness in security process enforcement, enabling isolation of areas of code developed during these periods for more rigorous testing and maintenance. This data would also allow the project to make informed decisions about when the technical debt of

such flaws had accumulated sufficiently that it was economically advantageous to refactor the affected portions of the code base.

For example, in other work we have shown how certain types of design flaws are consistently highly correlated with high bug and change rates [12, 13]. If these design flaws are not fixed, no amount of bug-finding and bug-fixing effort will result in higher quality software. We are interested to know whether the same patterns hold specifically for security bugs. If this were true then, armed with this information, a project manager could focus project resources on refactoring, to remove the design flaws, and hence systematically increase system security. But such an analysis is almost impossible today: most projects do not indicate which bugs in their issue-tracking system or which commits in their Version Control System (VCS) are security related. We simply lack the raw information to perform this analysis. To address this shortcoming we have focused on filling this gap—that is, providing the missing information—in software development project situational awareness. In this way we can provide insight to software architects and project managers as to the ongoing security efforts within their projects based on a data-driven assessment of their project repositories.

To achieve this goal, two common sources of incontrovertible data were identified for use: 1) text from tasks entered into trackers and 2) text messages accompanying source code commits to a VCS. It is expected that most organized software development teams use both issue trackers and some form of VCS that provides the opportunity for commit messages, and as such that these data sources will be readily available in the vast majority of operational contexts. To quantify security effort using these sources of data, natural language processing (NLP) methods were applied to derive feature sets from the unstructured text data and machine learning classification methods were applied to determine whether each text snippet was likely to represent a security-related topic. While others have attempted to use NLP techniques to detect specific security information within full text documents [14], our approach represents a generalized topic detection scheme with broad potential utility in informing software development processes.

II. BACKGROUND

NLP techniques have been used by researchers in a number of ways to analyze artifacts of the software development process, often focused on commit messages or defect reports [15, 16, 17]. Other researchers have noted the challenges and potential rewards of mining process and requirements data from software project artifacts [18, 19]. Our work builds on this research by attempting to create a generalized classifier capable of identifying security-focused text artifacts from data residing in standard software development tools.

The classification method that we have been exploring, Naïve Bayes, simplifies statistical learning processes by adopting an assumption of independence between features of a given classification. Though this assumption can be questioned for many applications, in practice Naïve Bayes competes well

against more sophisticated techniques, and is therefore a common starting point for exploration of a new problem space [20].

III. DATA

To train classifiers to identify software security-focused text snippets, reliable gold-standard data was required. The most accessible source of large quantities of such training data comes in the form of issue tracker items from open source projects in which contributors specifically label security-related issues with a “security” tag. The Chromium projects [21] hosted on the Google Code platform were identified as one such source. The issue-tracking repository for these projects included 875 issues labeled as “security”, dating from 3/10/13 to 1/9/15. The summary statements of these issues were extracted as positive examples of software security-related text snippets, while the summary statements of additional Chromium issues not tagged as “security” were used as negative examples. The full data set used in this study contained 1874 text samples (875 security related, 999 not security related). Table I shows examples of both positive and negative text snippets used for training.

Data were randomized and divided into training and testing sets for performance validation, discussed in detail in the next section.

IV. EXPERIMENTAL METHODS

This initial study applied Naïve Bayes classification to the problem of determining whether snippets of text are related to software security. To perform feature selection and classification, the Python programming language and the associated Natural Language Toolkit (NLTK) were used. Methods of Naïve Bayes classifier training, testing, accuracy calculation, and confusion matrix generation used were all unmodified NLTK implementations [22].

TABLE I. TRAINING DATA SAMPLES

Chromium Project Training Data	
<i>Summary Message (text snippet)</i>	<i>Correct Classification</i>
Clicking “Safe Browsing diagnostic page” link broken on malware interstitial	security
Block chrome-extension:// pages from importing script over non-HTTPS connections	security
Security: XSS issue in the FTP parser	security
Heap-use-after-free in WebCore::RenderLayer::repaintBlockSelectionGaps	security
Rendering glitch when switching windows	not security
Regression: Default cursor not seen in Sign in page of chrome after navigating back from any other tab.	not security
Status Bar fails to hide	not_security
Separators on column header disappear when display language is RTL.	not_security

A. Feature Extraction

Features for text classification were derived from the presence or absence of n-grams in tokenized text snippets. For example, the text snippet “I am a rock, I am an island” is tokenized into the following set of tokens: [‘i’, ‘am’, ‘a’, ‘rock’, ‘i’, ‘am’, ‘an’, ‘island’]. Prior to feature detection, stop words (words too common to indicate any semantic meaning for our classification) were removed. Removing stop words (including ‘i’, ‘am’, and ‘an’, as defined in [23]) from our example yields the following remaining tokens: [‘rock’, ‘island’]. These tokens indicate two unigram (or n-gram of size one) features for our text snippet, namely $contains(rock) = True$ and $contains(island) = True$. It is also relevant to know if a text snippet does *not* contain certain words, such as $contains(snowblower) = False$, as the presence of the word “snowblower” would likely impact the meaning or topic of our text. N-gram feature analysis often goes beyond unigram features to also include bigram (adjacent word pair) features [e.g., $contains(rock, island)$], trigram (adjacent word triplet) features, etc.

To determine the set of all potential features relevant to the domain of interest (i.e., software security), a training set of summary messages from issues in the open source Chromium project was analyzed, as depicted in Fig. 1.

Text from the summary message of each issue was tokenized and stop words were removed. The remaining tokens were used to generate feature spaces S_1 , S_2 , and S_3 , representing a unigram-only feature space, a (unigram + bigram) feature space, and a (unigram + bigram + trigram) feature space, respectively. These feature spaces were used to extract features from the text for classifier training and testing, as described in the next section.

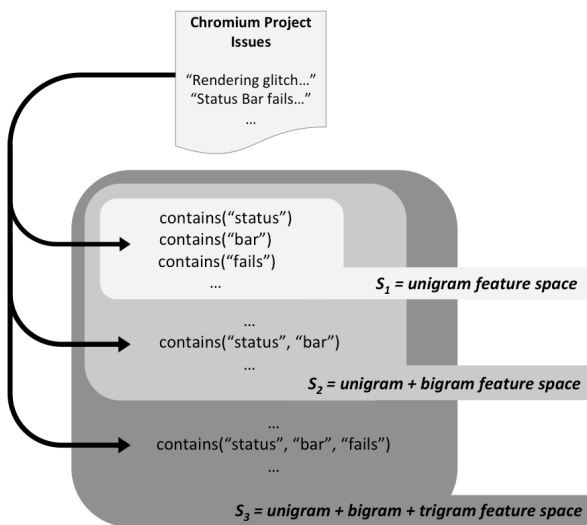


Figure 1: Generation of Experimental Feature Spaces

B. Naïve Bayes Classification

Once the full feature space was determined, text samples in a training set with known classifications could be analyzed to determine their feature vectors. For our text samples, a feature vector is a representation of the presence or absence of all features in the full feature space. Thus, when running an experiment using feature space S_1 , the feature vector $V_1(t)$ for a given tokenized text snippet t is an array containing a Boolean value for each token feature represented in S_1 , indicating whether the text contained or did not contain the token. See Fig. 2. These feature vectors and their correct classifications were used to train a Naïve Bayes classifier, yielding a statistical model of features and their statistical contributions to the classification of software security related text.

It should be expected that the words comprising highly informative features in models generated from training will be independent of words found by prior studies to be common to text snippets from many areas of software projects, such as those reported in [24]. For example, words like “html”, “add”, or “feature” would be common to almost any software project text, and thus would not be expected to yield highly informative features to our security classifier after training. Table II presents an example of the most informative features of a model from a single training run, illustrating unigram features and their likelihood ratios in classifying software security related text. True to expectations, the highly informative features discovered have no overlap with high-frequency words common across many software systems [24], lending confidence to the domain-centric training of the classifiers.

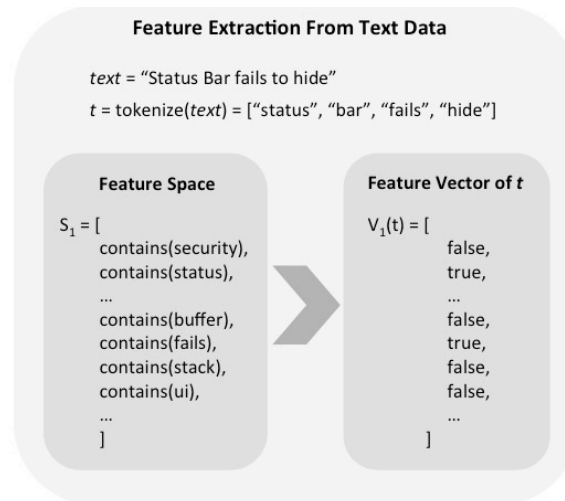


Figure 2. Feature Vector Generation

TABLE II. UNIGRAM CLASSIFIER MODEL EXAMPLE

Informative Features for Classifying Software Security Text	
<i>Feature</i>	<i>Likelihood Ratio (security : not_security)</i>
contains(heap) = True	25.3 : 1.0
contains(corruption) = True	18.1 : 1.0
contains(security) = True	17.9 : 1.0
contains(bad) = True	16.2 : 1.0
contains(integer) = True	15.2 : 1.0
contains(overflow) = True	13.9 : 1.0
contains(pointer) = True	12.0 : 1.0
contains(doesn) = True	1.0 : 11.4
contains(stack) = True	11.3 : 1.0
contains(buffer) = True	11.2 : 1.0
contains(seen) = True	1.0 : 10.5
contains(ui) = True	1.0 : 9.4

A number of words highly relevant to the topic of software security appear in the models derived from classifier training. As might be expected, we see that words such as “corruption”, “overflow”, and “security” are strong indicators that a text snippet should be classified as a software security message. Strong negative indicators also appeared, such as the presence of the word “ui” (user interface) indicating that the message was likely not related to software security. It is worth noting that while the top 15 indicators for each classifier training run were stored, no strong indicators based on the absence of a word were observed. This shows no indication, for example, that any single feature is so common in security-related text that its absence alone conveys strong semantic meaning.

Further experiments expanded the feature space by allowing bigram (two-word) and trigram (three-word) features to be used in classification. Table III shows an example feature model for a classifier derived using a feature space containing unigrams, bigrams, and trigrams.

It can be seen that new strong indicators were introduced by the addition of larger n-grams, including the (buffer, overflow) bigram and (heap, buffer, overflow) trigram. These phrases are consistent with expected terms highly relevant to software security, and provide positive anecdotal evidence for the classifier training data and methodology.

C. Validation

Repeated random sub-sampling validation [25] was performed to validate the approach to text classification. This validation method was chosen to demonstrate the efficacy of training functional classifiers from many possible selected data sets in the hopes of proving the approach robust without specifically selected training data. Repeated random sub-sampling is performed by repeatedly splitting gold standard data into two randomly distributed partitions of pre-defined

proportions, training and testing the classifier for each split, and recording all performance results.

TABLE III. UNIGRAM + BIGRAM + TRIGRAM CLASSIFIER MODEL EXAMPLE

Informative Features for Classifying Software Security Text	
<i>Feature</i>	<i>Likelihood Ratio (security : not_security)</i>
contains(heap) = True	38.7 : 1.0
contains(overflow) = True	27.3 : 1.0
contains(security) = True	24.9 : 1.0
contains(cast) = True	23.4 : 1.0
contains(pointer) = True	21.8 : 1.0
contains(bad) = True	18.3 : 1.0
contains(buffer, overflow) = True	17.8 : 1.0
contains(doesn) = True	1.0 : 17.7
contains(corruption) = True	17.3 : 1.0
contains(fails) = True	1.0 : 14.3
contains(integer) = True	13.9 : 1.0
contains(buffer) = True	12.6 : 1.0
contains(heap, buffer) = True	12.3 : 1.0
contains(heap, buffer, overflow) = True	12.3 : 1.0

In this study, three experiments were performed, using different feature sets. The first feature set included only unigram (single word) tokens observed in data from the Chromium project. The second experiment used a feature set containing unigrams and bigrams, and the third experiment used a feature set containing unigrams, bigrams, and trigrams. In each experiment, we performed 50 repetitions of random sub-sampling on our aforementioned issue tracker data from the Chromium project. The full experimental data set was distributed into 80/20% training/test distributions, resulting in random training sets containing 1499 samples and random test sets containing 375 samples.

Classifier performance from the experiments can be seen in Table IV.

TABLE IV. RESULTS OF N-GRAM FEATURE STUDIES

Classifier Performance for Various n-gram Feature Spaces			
<i>Feature Space</i>	<i>Average Precision</i>	<i>Average Recall</i>	<i>Average F-Measure</i>
S ₁ (Unigrams, 12674 total features)	0.91 ± 0.026	0.89 ± 0.023	0.90 ± 0.019
S ₂ (Unigrams + Bigrams, 25347 total features)	0.92 ± 0.022	0.88 ± 0.022	0.90 ± 0.015

Classifier Performance for Various n-gram Feature Spaces			
Feature Space	Average Precision	Average Recall	Average F-Measure
S ₃ (Unigrams + Bigrams + Trigrams, 38019 total features)	0.93 ± 0.017	0.88 ± 0.020	0.91 ± 0.016

Overall, the trained classifier performed well at software security topic detection. The average precision (or positive predictive value), which measures the fraction of text snippets classifier as “security” by our classifier that were proven to be correct classifications, was observed between 91% and 93% in our experiments. This data not only represents a promising classifier, but show that the addition of more complex features (bigrams and trigrams) increases performance of a classifier for this domain. Recall (or true positive rate), which measures the fraction of correctly classified security text snippets out of the total number of security text snippets in the data set, was measured at between 88% and 89% in our experiments. Finally, the average f-measure (the harmonic mean of precision and recall) increased from 90% to 91% as more (and more complex) features were added.

V. CONCLUSIONS

This paper has presented the results of an initial study investigating the potential of applying NLP and machine learning techniques to extract information from data residing in VCSs and issue tracking systems. The information that we extracted in this study was a classification of issues as security related or not security related. Using this information we can create measures of, and get insights into, software process and software quality.

We have shown here that we can fully automate the process of extracting semantically meaningful information from issue-tracking systems and that this information has both high precision and high recall. Our belief is that the precision and recall are high enough that this technique will open up many possibilities for post-hoc analysis of project repositories and communications, enabling insights that were hitherto impossible, due to the dearth of data. While our goal here was to identify security-related issues, we believe that this technique has the potential to “mine” many other kinds of data from project repositories.

It is expected that more sophisticated feature extraction and classification techniques may further improve on these results. The initial success of this methodology using token-derived features also indicates that the lexicon of security within software development is sufficiently common and consistent that this domain is ripe for the sort of analysis presented here. As training data was extracted, unaltered, from active open source projects with no interaction with developers generating the text, we remain confident that the language used naturally within this domain will yield successful training data from other software development projects in the future.

VI. FUTURE WORK AND LIMITATIONS

We recognize a number of limitations and areas for fruitful expansion of this work, including (1) a broader number of data sets, spanning a wider range of software project types and teams, (2) more sophisticated classification methods such as Support Vector Machines or ensemble methods, and (3) application of these techniques to detect software quality attributes other than security.

It is hoped that versions of the classifiers demonstrated here can be proven effective in classifying not only text snippets from tasks in issue-tracking systems, but also to classify other text snippets common to the software development lifecycle, such as commit messages in version control systems. Validation of this will be explored in future studies.

Additionally, the authors plan to explore the efficacy of this approach in autonomously measuring and monitoring a variety of software quality attributes from data derived from standard software process management and DevOps systems. For example, it would be valuable to monitor when the occurrences of issues related to other quality attributes—such as usability or availability or safety—was spiking.

It is anticipated that the achievement of autonomous quality monitoring can be leveraged into real-time alerting and predictive capabilities suitable for providing expert decision support to software development management, and proactively improving the quality of software developed by organizations employing the envisioned data-driven process optimization techniques.

ACKNOWLEDGMENT

We would like to acknowledge the support of Carol Woody, Bob Ellison, Yuanfang Cai, and Qiong Feng for this research. In addition we gratefully acknowledge the support of the U.S. Department of Homeland Security.

Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE

MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0002234

REFERENCES

- [1] Target Puts Data Breach Cost at \$148 Million, and Forecasts Profit Drop. *New York Times*. August 2014, retrieved March 2015. <http://www.nytimes.com/2014/08/06/business/target-puts-data-breach-costs-at-148-million.html>
- [2] JPMorgan Hack Exposed Data of 83 Million, Among Biggest Breaches in History. Reuters. October 2014, retrieved March 2015. <http://www.reuters.com/article/2014/10/03/us-jpmorgan-cybersecurity-idUSKCN0HR23T20141003>
- [3] U.S. Postal Service Says It Was Victim of Data Breach. *The Wall Street Journal*. November 2014, retrieved March 2015. <http://www.wsj.com/articles/u-s-postal-service-says-it-was-victim-of-data-breach-1415632126>
- [4] M. Howard and S. Lipner. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006.
- [5] The Heartbleed Bug, 2014. <http://heartbleed.com/>
- [6] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. ACM, New York, NY, USA, 475-488.
- [7] MITRE. 2014b. CVE-2014-7169. Common Vulnerabilities and Exposures. November 1, 2014: <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-7169>
- [8] OWASP, Comprehensive, lightweight application security process, <http://www.owasp.org>, 2006.
- [9] G. McGraw, *Software Security: Building Security*, Addison Wesley (2006).
- [10] Bart De Win, Riccardo Scandariato, Koen Buyens, Johan Grégoire, Wouter Joosen, On the secure software development process: CLASP, SDL and Touchpoints compared, *Information and Software Technology*, Volume 51, Issue 7, July 2009, Pages 1152-1171.
- [11] MITRE, Common Vulnerabilities and Exposures, <https://cve.mitre.org/>, 2015.
- [12] L. Xiao, Y. Cai, and R. Kazman, "Design Rule Spaces: A New Form of Architecture Insight", *Proceedings of the International Conference on Software Engineering (ICSE) 2014*, (Hyderabad, India), June 2014.
- [13] R. Mo, Y. Cai, R. Kazman, and L. Xiao, "Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells", *Proceedings of The Working IEEE/IFIP Conference on Software Architecture (WICSA 2015)*, (Montreal, Canada), May 2015, in press.
- [14] Xusheng Xiao, Amit Paradkar, Suresh Thummalapenta, and Tao Xie. 2012. Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12)*.
- [15] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. 2011. Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*. ACM, New York, NY, USA.
- [16] Runeson, P.; Alexandersson, M.; Nyholm, O., "Detection of Duplicate Defect Reports Using Natural Language Processing," *ICSE 2007. 29th International Conference on Software Engineering, 2007*, pp.499-510, 20-26 May 2007.
- [17] Rubén Prieto-Díaz. 1991. Implementing faceted classification for software reuse. *Communications of the ACM* 34, 5 (May 1991), 88-97.
- [18] Poncin, W.; Serebrenik, A.; van den Brand, M., "Process Mining Software Repositories," *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pp.5,14, 1-4 March 2011.
- [19] Cleland-Huang, J.; Settimi, R.; Xuchang Zou; Solc, P., "The Detection and Classification of Non-Functional Requirements with Application to Early Aspects," *Requirements Engineering, 14th IEEE International Conference*, pp. 39,48, 11-15, Sept. 2006.
- [20] Rish, Irina. "An empirical study of the naive Bayes classifier." *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. No. 22. IBM New York, 2001.
- [21] The Chromium Projects, 2015, <http://www.chromium.org/>
- [22] Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.
- [23] Porter, M. F. 1980. "An Algorithm for Suffix Stripping." *Program*, 14(3), 130-37.
- [24] Alali, A.; Kagdi, H.; Maletic, J.I., "What's a Typical Commit? A Characterization of Open Source Software Repositories," *International Conference on Program Comprehension (ICPC) 2008*, 10-13 June 2008, pp.182-191.
- [25] Ron Kohavi. 1995. "A study of cross-validation and bootstrap for accuracy estimation and model selection." In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'95)*, Vol. 2. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Towards Goal-Oriented Conformance Checking

Hiroki Horita, Hideaki Hirayama, Yasuyuki Tahara and Akihiko Ohsuga

Graduate School of Information Systems

The University of Electro-Communications

Tokyo, Japan

Email: h-horita@ohsuga.is.uec.ac.jp, hirayama968@ybb.ne.jp, {tahara,ohsuga}@is.uec.ac.jp

Abstract—Constructing a business process is important area between requirements engineering and business process management. Goal-oriented requirements analysis method is widely researched in requirements engineering and useful for reflecting organizational requirements to business process models, but actual business processes deviate from defined process models. Therefore, it is not sufficient for business process analysis only using model's information. It is important to analyze actual conducted business process logged data. Analyzing business process logged data is called process mining and detecting differences between models and logs is called conformance checking. A lot of conformance checking approaches mainly focus on process aspects of business process, but this is not sufficient for analysis whether actual business processes can satisfy organizational goals. In this paper, we propose a goal-oriented conformance checking approach which can detect deviations between logs and models, and can analyze the effects of the deviation. It is useful for evaluation of the detected deviation. We represent the effectiveness of our approach conducting a case study using the publicly available log.

I. INTRODUCTION

In recent years, problems of business process complexity and rapidly changing business environments are needed to deal with. In that situation, constructing business process is used for discussion, verification, documentation and etc [1]. Using goal models are effective against constructing appropriate business process models. Goal models are researched in requirements engineering area, and used for requirements analysis [2]. Goal models have systematic and logical construction for representing requirements for information systems. These characteristics are useful for reflecting organizational requirements to business process models.

Using goal models for constructing business process models is effective, but it is not sufficient for organization, because actual business often deviate from defined business process models [1]. In that situation, model-based analysis is not adequately effective. Therefore, in recent years, analyzing business process logged data is widely researched and it is called process mining. In process mining, conformance checking between models and logs are important topics. Conformance checking can detect deviations between normative models and actual logged data (it is called event logs). When the models and reality (logged data) have little in common, model-based analysis does not make much sense [1]. Therefore, analyzing logs and improving models are important.

Quite a lot of conformance checking researches are conducted. At first, these approaches only focus on control-flow perspective afterward, data (event related information) and resource (agent conducting the event) perspectives are

focused. These approaches can check various perspectives relating business process. These perspectives are important, but it is not sufficient yet. Using these approaches against logs and models represents deviations between them, but it is not concrete what should we do for interpreting deviation and improving business process.

In this work, we propose a goal-oriented conformance checking approach. Goal-oriented aspects are effective against deviation interpretation. Goal models have systematic and logical construction. Therefore, it is possible to represent what is important in a business process and encourage efficient decision making. Our approach is constructed by two phases. The first phase checks deviation between a goal model and logs focusing control-flow, data and resources. Goal models are described using linear temporal logic, so verification is formally conducted. In addition, in case of deviation detected, goals are combined with a goal using goal model construction for evaluating the deviation. Therefore, a cross tabulation table relating these goals is constructed. In the second phase, we calculate significant difference between two goals in a cross tabulation table. Next, if these goals have significant differences, a relation between two goals are positive or negative is calculated. In this way, deviations are detected and evaluated the effects of the deviation. It is useful for interpretation of deviations between processes and logs.

II. GOAL MODEL AND BUSINESS PROCESS MANAGEMENT

Goal models are used for requirements elicitation, evaluation, negotiation, elaboration, structuring, documentation, analysis and evolution [2] for system development. Goals should be achieved and refined into subgoals through AND/OR decompositions.

Relating goal models and business process models are important. Various researches are conducted in this area (e.g. transformation:[3],[4], validation: [5], integration: [6]). These approaches are mainly used in business process construction phase. It is corresponding to diagnosis/requirements and (re)design phases in the business process life cycle [1]. These phases are important, but business processes are life cycle and improved at various times. Therefore, it is needed to confirm desirable business process that can achieve organizational goals. In many cases, actual processes deviate from the normative business process model. In these cases, it is difficult to know the effect of deviations from business process models using only model information because models only have normative information. Therefore, it is needed to use event logs of business process and to confirm event logs can

achieve business goal or not. These are enactment/monitoring and adjustment phase in business process life cycle. It is needed to research using goal models, business process models and event logs for continual improvement of business process.

III. APPROACH

In this section, we present the details of the proposed approach. Figure 1 sketches the proposed Goal-oriented conformance checking method. It relies on two phases: Trace & Goal Processing phase to filter traces by a goal is achieved or not and Combine Goals to construct a cross tabulation table and a Statistical Analysis phase to measure significant differences between two goals in a cross tabulation table constructed in prior phase and evaluate the deviation have positive effects or negative effects.

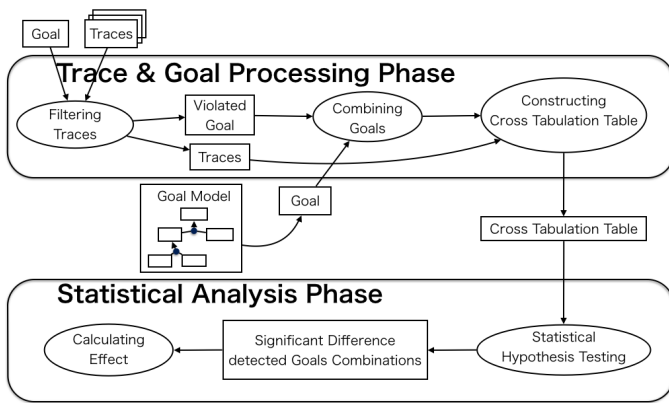


Fig.1. Overview of Our Proposed Method

A. Trace & Goal Processing phase

Trace & Goal Processing phase contains three processes and represented as an ellipse in Trace & Goal Processing phase of Figure 1. This phase conducts checking whether event logs satisfy goals of a goal model represented as logical formula and a constructing cross tabulation table for the next phase.

First step, Filtering Traces uses a goal of a goal model and traces as inputs and divides all traces to a goal satisfied traces or not satisfied traces. Goals are described using linear temporal logic, so it is possible to verify the trace satisfy the goal or not using LTL checker [7] on ProM. If any of these traces can not satisfy the goal, the goal is considered as violated goal. Second step, Combining Goals conduct combining the violated goal and a more upper or a high priority goal using information about a goal model configuration. more upper or high priority goals is more important than low level goals. Combining Goals is conducted for evaluating the violated goal influences other goals or not. Third step, Constructing Cross Tabulation Table use goals combined in the prior step and use these goals to construct a cross tabulation table. Table I is a cross tabulation table we want to construct. The table has two variables which represent a each combined goal is achieved or not. B in Table I represents a case when the violated goal is achieved, conversely, !B in Table I represents a case when the violated goal is not achieved. A in Table I represents a case when the combined goal is achieved, conversely, !A in Table I represents a case when the combined goal is not

achieved. Therefore the cross tabulation table has 2×2 cells which represent traces numbers of $(A \wedge B)$, traces numbers of $(A \wedge !B)$, traces numbers of $(!A \wedge B)$, traces numbers of $(!A \wedge !B)$. The cross tabulation table represents the correlation between goal A and B. These numbers in cells are used in next statistical analysis phase. In these ways, the deviation from process defined by goal models are detected and preparation for the next phase are conducted.

TABLE I. CROSS-TABULATION TABLE USED FOR STATISTICAL HYPOTHESIS TESTING

	B	!B
A	trace numbers of $(A \wedge B)$	trace numbers of $(A \wedge !B)$
!A	trace numbers of $(!A \wedge B)$	trace numbers of $(!A \wedge !B)$

B. Statistical Analysis phase

Statistical Analysis phase contains two processes and represented as an ellipse in the Statistical Analysis phase of Figure 1. This phase analyzes whether goals achievement relation have significant differences and when the goal is not achieved, whether it have positive effects or negative effects against achieving the combined goal are evaluated.

First step, Statistical Hypothesis Testing uses cross tabulation table constructed in the prior phase as an input. We use Chi-squared test and Fisher's exact test. These statistical hypothesis testing methods are suitable for cross tabulation tables including categorical data and used for testing independence between two variables. If goal relations have significant differences in the significance level of 0.05 ($p\text{-value} < 0.05$), we consider these goals having relations. If some cells have lower values, Fisher's exact test is used. Second step, Calculating Effect are conducted for goals having significant differences in the significance level of 0.05 ($p\text{-value} < 0.05$). In this step, it is evaluated that violated goal influences positive effects or negative effects in a combined goal. Therefore, we use below equation (1). A is the number of traces when a combined goal is achieved. B is the number of traces when a violated goal is achieved. !A and !B are the number of traces when a each goal is not achieved. The equation represents the effect of not achieving goal B concerning goal A. If the value of *effect* is positive, not achieving goal B concerning goal A has positive relation. If value of *effect* is negative, not achieving goal B concerning goal A has negative relation. In this way, a relation between two goals are evaluated.

$$effect = \frac{A \wedge !B}{A} - \frac{!A \wedge B}{!A} \quad (-1 \leq effect \leq 1) \quad (1)$$

IV. CASE STUDY

We have evaluated our approach on an event log which taken from a phone repair process and is publicly available and used in some researches for evaluations. The log contained 11855 events from 12 different events in 1104 cases, each case representing a phone terminal repair process (register, analyze defect, repair, test repair, archive and etc.). The log data format is XES. Each trace describes a sequential list of events corresponding to a particular case. The log, its traces, and its events may have any number of attributes [1]. Attributes are standard (case id, time and etc.) or domain specific (phone

TABLE II. GOALS AND FORMALLY DEFINED GOALS OF GOAL MODEL

goal name	formal defined goal	goal type	priority
Achieve [repairing phone]	$\diamond \text{Repair (S)} \vee \diamond \text{Repair (C)}$	event	low
Achieve[archiving information]	$\diamond (\text{Archive Repair})$	event	low
low repair numbers	$\text{numberRepairs} < 3$	constraint	medium

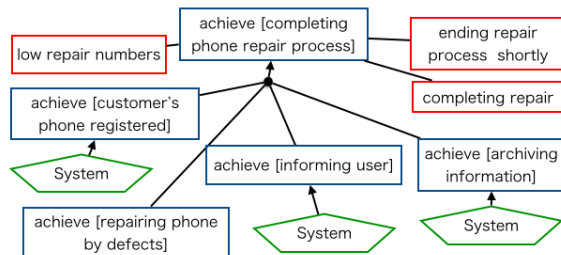


Fig.4. partial goal model of phone repair process

type, defect type and etc.). We constructed a goal model and use the model in Figure 4 for evaluations. Blue rectangles represent event and process related goals. Red rectangles represent constraint goals. Green pentagon represents agent that should achieve the goal devoted to the arrow. These goals are partially and formally described in Table II using linear temporal logic.

A. Goal: low repair numbers & other goals

In this section, we explain cases when goal: low repair numbers are detected as a deviation. First, in Trace & Goal Processing phase, goal: low repair numbers is combined with goal: completing repair using goal model construction. Next, a cross tabulation table is constructed. This is represented in the table III. This table represents 4 cases which show the number of both goals are achieved, only one goal is achieved and both goals are not achieved. Next, Statistical analysis phase uses this cross tabulation table. Fisher's exact test is used for calculating significant differences between these goals and p-value are calculated. The results are described in table IV. Calculated p-value is $2.2e^{-16}$, so this has significant differences in significance level of 0.05. Therefore, next, the effect is calculated using equation (1). The equation can represent that not achieving goal: low repair numbers is positive effects or negative effects against Goal: completing repair. The result is represented in table IV. Effect value is -0.903 (truncate a number to 3 decimal places). Therefore, not achieving Goal: low repair numbers detected as deviations are negative effects against Goal: phone repair completed.

TABLE III. CROSS-TABULATION TABLE OF GOAL (COMPLETING REPAIR) & GOAL (LOW REPAIR NUMBERS)

	low repair numbers	!(low repair numbers)
completing repair	1014	50
!(completing repair)	2	38

V. RELATED WORK

In this section, we explain related work about conformance checking. Related works excepted for conformance checking are lined up in section 2. Rozinat et al proposed token replay conformance checking method [8]. It measures the fitness

TABLE IV. P-VALUE & EFFECT

A: combined goal & B: violated goal	testing method	p-value	effect
A: completing repair & B: low repair numbers	Fisher's exact test	$2.2e^{-16}$	-0.903
A: ending repair process shortly & B: low repair numbers	Pearson's Chi-squared test	$2.2e^{-16}$	-0.317

of the process model and event log. Adriansyah proposed alignment based conformance checking method [9]. It makes it possible to check conformance more precise. Leoni et al proposed multi perspective conformance checking method [10]. This technique deal with control flow, data and resource for conformance checking.

These researches mainly focus on deviation between process models and event logs. This is an important aspect in process improvements, but detecting deviation between process models and event logs are not conclusive destination. Utilizing the results of conformance checking as a means to improve business process models, business rules and to reconfigure business goals should be conducted. In this perspective, our proposed conformance checking method is suitable for this.

VI. CONCLUSION

Today's organizations need to comply with a rapidly changing business environments and need to set goals against the change. In this paper we proposed a goal-oriented conformance checking method. The method can detect deviations between goal models and logs using the verification method based on linear temporal logic and can evaluate detected deviations using statistical analysis. It is useful for constructing more appropriate business processes and organizational goals.

REFERENCES

- [1] W.M.P. van der Aalst. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Verlag, 2011.
- [2] A. van Lamsweerde. Requirements Engineering :From System Goals to UML models to software specifications. WILEY, 2009.
- [3] H. Horita, K. Honda, Y. Sei, H. Nakagawa, Y. Tahara and A. Ohsuga " Transformation Approach from KAOS Goal Models to BPMN Models Using Refinement Patterns," in SAC 2014, ACM, Gyeongju, pp. 1023-1024, 2014.
- [4] J.L. de la vara, J. Sánchez, and Ó. Pastor. "On the Use of Goal Models and Business Process Models for Elicitation of System Requirements," in BPMDS 2013, Springer, Valencia, pp.168-181, 2013.
- [5] G. Gröner, M. Asadi, B. Mohabbati, D. Gašević and F. Silva Parreiras and Marko Bošković. "Validation of User Intentions in Process Models," in CAiSE 2012, Springer, Berlin, pp. 366-381, 2012.
- [6] M. Ruiz, D. Costal, S. España, X. Franch, Ó. Pastor, "Integrating the goal and business process perspectives in information system analysis," in CAiSE 2014, Springer, Heidelberg, pp. 332-346, 2014.
- [7] W.M.P. van der Aalst, H. de Beer and B. van Dongen, "Process Mining and Verification of Properties: An Approach Based on Temporal Logic," in OTM 2005, Heidelberg, pp. 130-147, 2005.
- [8] A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems, 33 : pp. 64-95, March 2008.
- [9] A. Adriansyah, "Aligning Observed and Modeled Behavior", PhD Thesis. Technische Universiteit Eindhoven, The Netherlands, 2014.
- [10] M. de Leoni and W.M.P. van der Aalst, "Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming," in BPM 2013, Splinger, pp. 113-129, Beijing, 2013.

Image retrieval based on structural and textual context

Sana FAKHFAKH
Laboratory MIRACL,
Institute of Computer Science
and Multimedia of Sfax,
Sfax University, Tunisia
Email: sanafakhfakh@yahoo.fr

Mohamed TMAR
Laboratory MIRACL,
Institute of Computer Science
and Multimedia of Sfax,
Sfax University, Tunisia
Email: mohamed.tmar@isimsf.rnu.tn

Walid MAHDI
Laboratory MIRACL,
Institute of Computer Science
and Multimedia of Sfax,
Sfax University, Tunisia
Email: walid.mahdi@isimsf.rnu.tn

Abstract—In this paper, We propose a geometric method who use implicitly of textual and structural context of XML elements and we are particularly interested by improve the effectiveness of various structural factors for multimedia retrieval. Using a geometric metric, we can represent structural information in XML document with a vector for each element. Experimental evaluation is carried out using the INEX 2007, ImageCLEF 2010 and 2014. The results show that integration of structural context significantly improves compared results of using a single textual context.

Keywords—Structural context, Textual context, Approximative resolution, XML element, Image retrieval

I. INTRODUCTION

In this article, we focus on techniques for multimedia retrieval based on textual and structural context in XML documents. This type of document includes textual information and structural constraints. So, XML document cannot be effectively exploited by classical techniques of information retrieval, which regard document as a plane source of information. The implicit incorporation of multimedia elements in XML documents requires the exploitation of textual context for multimedia retrieval. However, the textual context remains insufficient in most of time. The idea is to calculate the relevancy score of media element based on information from the textual and structural context to answer a specific information needs of user, expressed as query composed of set of keywords. Our main inspiration is to use the structure to involve each textual information depending on its position in XML document, that is textual information that gives the best possible description of multimedia element. In our work, we will be interested by media "image".

II. PROPOSED APPROACH

We propose a new metric for multimedia retrieval in XML documents which involves the use of geometric distances to calculate the relevance of each node from the multimedia node. This method consists of placing the nodes of XML document in Euclidean space and define each node by a vector of coordinates to calculate then the distance between each pair of nodes. This distance will play a beneficial role in to calculate the score of multimedia element. Thus, it facilitates the presentation of information in terms of interpretation and exploitation. Replaying to this need, we propose a new method in the field

of multimedia retrieval that takes into account the structure as a source of evidence and its impact on search performance. We present a new source of evidence dedicated to multimedia retrieval based on the intuition that each textual node contains information that describes semantically a multimedia element. And the participation of each text node in the score of a multimedia element varies with its position in there XML document. To compute the geometric distance, we initially place the nodes of each XML document in an Euclidean space to calculate the coordinates of each node by a detailed algorithm in our paper [1]. Then, we compute the score of a multimedia element depending on the distance between each textual node. We evaluate our system into three databases extracted from three collections : INEX 2007 (Initiative for the Evaluation of XML Retrieval) Ad Hoc task, ImageCLEF 2010 Wikipedia image retrieval task and ImageCLEF 2014 Plant task. The first two databases are composed by XML documents extracted from Wikipedia. The latest dataset is collected by scientific community for testing and validation of their approaches.

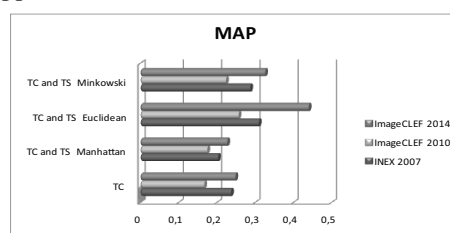


Fig. 1. Results of impact our approach on INEX 2007, ImageCLEF 2010 and ImageCLEF 2014 based in MAP(Mean Average Precision).

III. CONCLUSION

In this work, we studied the impact of textual and structural context on multimedia element retrieval, where the user need can be a multimedia element (text). We plan to investigate the impact of a mixture of text and multimedia element (text + image) with to using visual descriptors.

REFERENCES

- [1] S. Fakhfakh, M. Tmar, and W. Mahdi, "Multimedia retrieval based on geometric distance in semi-structured document," in *WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies, Barcelona, Spain, 2014*, pp. 220–225.

Probabilistic Failure-causing Schema in Input-Domain Testing

Ziyuan Wang Yuanchao Qi Jiawei Lin

School of Computer, Nanjing University of Posts and Telecommunications, Nanjing, 210003, China

Email: wangziyuan@njupt.edu.cn

Abstract—To describe characteristics of failure test cases in the input-domain testing, we propose a model of probabilistic failure-causing schema. In this model, test case that contains a probabilistic failure-causing schema has a probability to be a failure test case. It may help testers to find out input characteristics that have more close relationship to the fault.

I. INTRODUCTION

Once there are failure test cases reported in input-domain testing, input-level fault localization technique aims to find out characteristics of failure test cases.

To describe the characteristics, a model of minimal failure-causing schema was proposed [1]. Considering a boolean expression: $a \wedge (\neg b \vee \neg c) \wedge d \vee e$, and a clause disjunction fault (CDF) mutant: $a \wedge (\neg b \vee \neg c) \wedge d \vee (d \vee e)$ [2]. There are total 5 failure test cases. In all 7 failure-causing schemas, (- 1 1 1 0) and (0 - - 1 0) are minimal ones. They predict that all 5 input variables are involved in the fault. But factually, only d and e are related to this fault, where e is replaced by $d \vee e$.

	a	b	c	d	e
$test_1$	0	0	0	1	0
$test_2$	0	0	1	1	0
$test_3$	0	1	0	1	0
$test_4$	0	1	1	1	0
$test_5$	1	1	1	1	0

	a	b	c	d	e
$schema_1$	0	0	0	1	0
$schema_2$	0	0	1	1	0
$schema_3$	0	1	0	1	0
$schema_4$	0	1	1	1	0
$schema_5$	1	1	1	1	0
$schema_6$	-	1	1	1	0
$schema_7$	0	-	-	1	0

In this paper, we propose a model of probabilistic failure-causing schema to describe this phenomena.

II. PROBABILISTIC FAILURE-CAUSING SCHEMA

Considering a program under test with n input variables, each variable has a value set V_i ($i = 1, 2, \dots, n$). The input domain of program is $D = V_1 \times V_2 \times \dots \times V_n$.

Definition 1 (test case). A test case is a n -tuple ($v_1 \in V_1, v_2 \in V_2, \dots, v_n \in V_n$).

Definition 2 (schema). A k -value schema (or called a schema with strength k) s is a n -tuple ($-, \dots, -, v_{i,1}, -, \dots, -, v_{i,2}, -, \dots, -, v_{i,k}, -, \dots, -$) where $1 \leq k \leq n$. Where, the values of k variables have been fixes, while the values of other $n - k$ variables have not been fixed as denoted as "-".

Definition 5 (probabilistic failure-causing schema). A k -value schema is a k -value probabilistic failure-causing schema with a failure probability p_{fail} , if the ratio of the number of failure test cases that contain such schema to the number of test cases that contain such schema is p_{fail} .

A schema with failure probability p_{fail} means that, for arbitrary test case $t \in D = V_1 \times V_2 \times \dots \times V_n$ that contains such schema, the probability that t is a failure one is p_{fail} .

Definition 6 (coverage probability). The coverage probability p_{cov} of a schema is the ratio of the number of test cases that contain such schema to the number of all test cases.

A schema with coverage probability p_{cov} means that, for arbitrary test case $t \in D = V_1 \times V_2 \times \dots \times V_n$, the probability that t contains such schema is p_{cov} . There is negative correlation between the coverage probability and the strength of schema

III. APPROACH

People need characterize schemas with both higher failure probability and higher coverage probability, since the more p_{fail} means there is closer relationship between the schema and the fault, and the the more p_{cov} predicts less input variables that may be concerned with fault. By define a metric:

$$score(s) = p_{cov}(s) \times p_{fail}(s)$$

We can select probabilistic failure-causing schemas with the greatest score in input-level fault localization.

For previous example, we calculate p_{fail} and p_{cov} for each schema in 5 failure test cases (see Fig. 1). Three probabilistic failure-causing schemas with the greatest scores include:

$$\begin{aligned} score(- - - 1 0) &= \frac{1}{4} \times \frac{5}{8} = \frac{5}{32} \\ score(- - - 1 -) &= \frac{1}{2} \times \frac{5}{16} = \frac{5}{32} \\ score(- - - - 0) &= \frac{1}{2} \times \frac{5}{16} = \frac{5}{32} \end{aligned}$$

They predict that 2 variables d and e are involved in the fault.

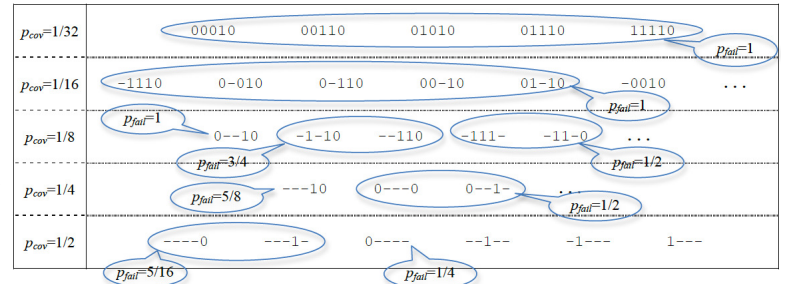


Fig. 1. Probabilistic failure-causing schemas (only some with greate p_{fail})

IV. CONCLUSION

A model of probabilistic failure-causing schema in input-domain testing is proposed. A simple example shows that they could reveal the source of faults more precision. More experiments are required in future works.

REFERENCES

- [1] C. Nie, H. Leung. The Minimal Failure-causing Schema of Combinatorial Testing. ACM Transactions on Software Engineering and Methodology (TOSEM), 2011, 20(4): 15.
- [2] Z. Chen, T. Y. Chen, B. Xu. A Revisit of Fault Class Hierarchies in General Boolean Specifications. ACM Transactions on Software Engineering Methodology (TOSEM), 2011, 20(3).

CARE: A Computer-Aided Requirements Engineering Tool for Problem-Oriented Software Development

Guoyuan Liu

College of Computer Science and Information Technology
Guangxi Normal University
No. 15 Yu Cai Road, Guilin,
Guangxi 541004, China
153123439@qq.com

Zhi Li*

College of Computer Science and Information Technology
Guangxi Normal University
No. 15 Yu Cai Road, Guilin,
Guangxi 541004, China
zhili@gxnu.edu.cn

Zhaofeng Ouyang

College of Computer Science and Information Technology
Guangxi Normal University
No. 15 Yu Cai Road, Guilin,
Guangxi 541004, China
751194151@qq.com

Abstract—This paper presents a tool to help software design in the development process. This software prototype will promote further development of Problem Frames framework (PF) and drive it to maturity, i.e., from theoretical research to practical applications.

Keywords—Problem Frames (PF); Problem diagram; Computer-Aided Requirements Engineering (CARE)

I. INTRODUCTION

Software requirements engineering plays an important role in software development projects. So how to conduct the practice of requirements elicitation, modeling, analysis and transform the results into correct software specifications is a key factor contributing to the successes of software development projects. We designed and implemented a prototype based on the theoretical foundations and principles of a problem-oriented requirements modeling framework—Jackson’s Problem Frames approach [1,2] (PF for short). PF has been regarded as one of the major requirements engineering approaches for assisting system analysts in structuring software development problems. They deploy problem diagrams for capturing and describing important contextual information for the software solutions to be built.

Over the years, there have been many extensions and advancements in Problem Frames research. For example, Hall *et al* have proposed Problem-Oriented Software Engineering (POSE) as a theoretic framework for software development [3,4]. Other researchers have made many theoretical extensions to PF and applied them to requirements analysis and reasoning for safety-critical systems [5,6], and identifying reliability concerns [7]. However, how to embed the PF framework in a software development practice remains an open problem.

In this paper, we present a computer-aided requirements engineering (CARE) tool for system analysts to use in the requirements analysis phase of software development. This work is motivated by the challenges and difficulties faced by many software development practitioners when communicating, modeling, analyzing and elaborating requirements in the early phase of a software development project. The tool not only can animate a visual transformation of requirements models, but also provide a vehicle for an automated textual transformation of requirements statement, accordingly.

II. PROBLEM MODELLING AND ITS TRANSFORMATION RULES

The PF is further development of Jackson’s work on JSP (Jackson Structured Programming) and JSD (Jackson System Development) [9]. Its basic tenet is that in requirements analysis phase, we should first understand the contextual environment in which the problem occurs, before giving any software solution. The rationale behind it is that most modern software systems inevitably interact with their surrounding environment to serve their ultimate purposes – satisfying the problem owner’s needs. PF deploys problem diagrams – a visual modeling notation as a way of concretizing the problem owner’s needs or wishes into observable or measurable phenomena [2]. The following is a typical problem diagram describing an insulin injection control system for diabetes.

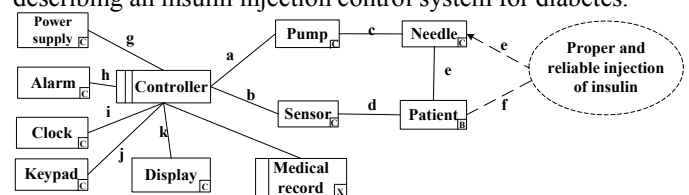


Fig. 1 Insulin injection control system for diabetes

Figure 1 shows that problem diagrams are extended context diagrams, with the following extensions:

- Rectangles with double stripes represent the computerized machine domain on which the software runs, e.g. the Controller domain;
- Application domains are represented by rectangles, which represent physical equipment (e.g., the Pump

*Zhi Li is the corresponding author. The research was supported in part by the Natural Science Foundation of China under Grant No.61262004, and the authors’ joint research is sponsored by the National Science Foundation of Guangxi Province under Grant No.2012GXNSFCA053010, and the Guangxi Scientific Research and Technological Development Project under Contract No.(Gui-Ke-He)1347004-22. Zhaofeng Ouyang is sponsored by University Innovation and Startup Project (No. 201410602104).

domain and the Sensor domain - the sub-labels with the symbol “C” represent “causal”, which means their properties or behaviors are predictable); or living beings (usually people, e.g., the Patient domain – the sub-label with the symbol “B” represents “biddable”, which means the domain has his own freewill but can follow orders or pre-determined rules after being trained or notified);

- Rectangles with a single stripe represent domains which can store information, e.g., the Medical record domain - the sub-label with the symbol “X” represents “lexical”, for instance, USB disks or other data storing devices;
- The solid lines labeled “a”, “b”, “c”, “d”, “e”, “f”, “g”, “h”, “i”, “j”, “k”, “l”, represent observable or measurable phenomena shared between domains;
- The dotted oval labeled “Proper and reliable injection of insulin” in Figure 1 represents the requirements. The text is a statement of needs or wishes of the problem owner (the diabetic patient in this case). The application domains that the statement concerns are connected with the oval by dotted lines – the Needle domain and the Patient domain; the label “e” and “f” represent the observable or measurable phenomena (either internal to the domains or external phenomena of the Needle and Sensor domain shared with other domains);
- The dotted rectangle which is connected with the Patient domain represents the Patient domain’s properties, i.e, “f” represents the phenomena “the patient’s blood sugar reaches abnormal level”, “d” represents the phenomena “the sensor detects the patient’s blood sugar reaches critical threshold level”, then f->d represents a cause-and-effect relationship that is the property of the Patient domain

From Figure 1, we can observe that PF represents a broad perspective on software development problems, in which the hardware, software and relevant application domains should all be treated as first-class citizens in the modeling process. Solving this kind of problems is a process of reasoning and moving from the dotted oval and the dotted lines towards the controller machine domain. In PF modeling, this process is known as problem transformation. In order to implement this transformation, we have defined three classes of transformation rules, namely the “cause-and-effect substitution rules”, “switching [domain’s] perspective rules”, and “removing [unconnected] domain rules”, see [8] for more details.

The contribution of this paper is that we have developed a computer-aided tool to implement the three classes of rules for problem transformation.

- The “cause-and-effect substitution” rule: since the application domain’s properties in PF modeling mainly describe causal-and-effect relationships in a form like “a->b”, we can substitute “cause” events with “effect” events or vice versa. The algorithm of this rule can be described by the following pseudo-code (variables are in italics):

```
foreach (i in D.event) //search all events of Domain D
  if (i == a) //find the event a
```

```
foreach (j in R.event) //search all events of requirement R
  if ((a -> b) is in D.property) //if a->b belongs to D’s property
    R.event [j]=a; //substitute a for b
```

- The “switching perspectives” rule: since adjacent domains share exactly the same set of phenomena, the requirement statement involving the shared phenomena can be switched from the viewpoint of the receiving end to the sending end, and vice versa. The algorithm of this rule can be described by the following pseudo-code:

```
foreach (i in D.event) //search all events from domain D
  foreach (j in R.event) //search all events of requirements R
    if (i == j) //if a equals b
      { D’.lines++; //add a new line to domain D’ of R
        D.lines--; //delete the old line connected to domain D
      }
```

- The “removing domain” rule: after the requirements are connected with the computing machine domain, all references and constraints of the requirements are on the computing machine, therefore, a requirements engineering problem becomes a pure programming problem, thus all other diagrammatic elements can be deleted. The algorithm of this rule can be described by the following pseudo-code:

```
if (R.ID is in Controller S.connectedID) //if requirement R is connected to the
//Controller
for (int i=0; i < Domain.count; i++) //search all the domains
  delete (Domain[i]); //delete domain
for (int i = 0; i > Model.count; i++) //search all the models
  delete (Model[i]); //delete model
```

III. TOOL OVERVIEW AND IMPLEMENTATION

A. Tool Overview

Figure 2 shows the overview of CARE, which consists of three main modules: the Application Domain module, the Transformation module and the Requirement module.

By combining the three modules we can draw a full problem diagram, in order to system analysts in requirements elicitation and analysis. More importantly, the transformation module can enable diagrammatic transformation, which is the core innovative part of the tool. So a computer-aided requirements engineering (CARE) prototype system has been designed. Its aim is to try to get prospective user involved in the process of requirements elicitation, modeling, analysis and transformation as early as possible and as visually engaging as possible. Another motivation for developing the prototype is to empirically evaluate the feasibility and practicality of the PF modeling and transformation framework.

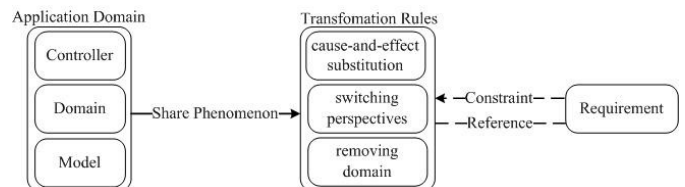


Fig.2 Design overview of CARE

B. Implementation

Since symbols and diagrams can be recognized at a glance and quite often, good representation can assist intuitive understanding of the meanings of visual and diagrammatic modeling. In addition, they can highlight some complex and important relationships among different entities without verbose or ambiguous texts.

Our prototype tool is designed to facilitate system analysts in drawing and editing problem diagrams, inputting a semi-structured textual statement of requirements, as a way of modeling requirements and relevant contexts. In addition, the tool also allows for visual transformation of the model and a textual transformation of requirements, which we believe can simulate or animate system engineers' process of reasoning while trying to solve engineering problems. From a requirements engineering perspective, this tool supports retaining requirements traceability, which is essential for requirements engineering [10]. The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations. fig.3 shows the interface of our CARE.

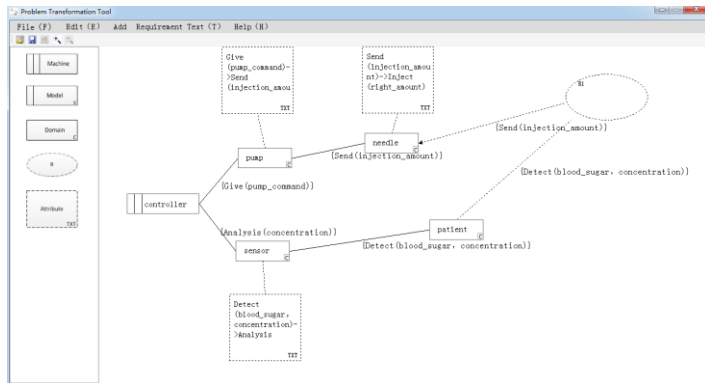


Fig.3 CARE: the Computer-Aided Requirement Engineering tool

IV. EVALUATION SETUP

We evaluated the usability of the CARE tool under both MS Windows and Android with many practical examples. We also report on the results of an initial empirical evaluation of the approach based on the prototype problem transformation tool. A total of 47 students took part in the evaluation, and the results are shown in table I.

TABLE I. PARTICIPANTS' ANSWERS TO THE QUESTIONNAIRE

How helpful is the tool to you in understanding Problem Frames?			
Extremely helpful	very helpful	somewhat helpful	not helpful
15	20	7	5

V. CONCLUSION

In this paper, we present a computer-aided requirements engineering tool. This is part of the second authors' long-term research towards moving PF closer to practice [8,12,13]. Currently, an early version of the prototype can be downloaded from the website <http://www.se.gxnu.edu.cn/tooldemo>. There are also several versions of the tool on mobile platforms, for example, an Android version and a Windows Phone version of the tool are also available on the website. When all tools are matured enough, we plan to empirically evaluate them by embedding PF theory and the CARE tools in realistic software development projects.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for the valuable comments, which helps improve this paper.

REFERENCES

- [1] Jackson M. Software requirements and specifications: a lexicon of principles, practices and prejudices [M]. Boston: Addison-Wesley, 1995.
- [2] Jackson M. Problem frames: analyzing and structuring software development problems [M]. Boston: Addison-Wesley, 2001.
- [3] Hall G H, Rapanotti L, Jackson M. Problem-oriented software engineering: a design-theoretic framework for software engineering, 2007[C]//Proceedings of the 5th IEEE International Conference on Software Engineering and Formal Methods. Los Alamitos: IEEE CS Press, 2007: 15-24.
- [4] Hall G H, Rapanotti L, Jackson M. Problem-oriented software engineering: solving the package router control problem[J]. IEEE Transactions on Software Engineering, 2008, 34(2): 226-241.
- [5] Strunk E A, Knight J C. The essential synthesis of problem frames and assurance cases[J]. Expert Systems, 2008, 25(1): 9-27.
- [6] Mannering D, Hall J G, Rapanotti L. Towards normal design for safety-critical systems[C]//Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2007: 398-411.
- [7] Yin B, Jin Z, Li Z. Reliability concerns in the Problem Frames Approach and system reliability enhancement patterns[J]. Jisuanji Xuebao (Chinese Journal of Computers), 2013, 36(1): 74-87.
- [8] Li Z, Hall J G, Rapanotti L. On the systematic transformation of requirements to specifications[J]. Requirements Engineering, 2013, (doi: 10.1007/s00766-013-0173-8), online first article.
- [9] Berry M D. Software requirements and design: the work of Michael Jackson[J]. ACM SIGSOFT Software Engineering Notes, 2011, 36(2): 39-40.
- [10] Jane Cleland-Huang, Orlena Gotel, Jane Huffman Hayes, Patrick Mäder, Andrea Zisman. Software traceability: trends and future directions. FOSE 2014: 55-69
- [11] Sommerville I. Software Engineering 9th Edition[M]. Boston: Addison-Wesley, 2011.
- [12] Rapanotti L, Hall G J, Li Z. Deriving specifications from requirements through problem reduction[J]. Journal of IEE Proceedings-Software, 2006, 153(5): 183-198.
- [13] Li Z, Hall G J, Rapanotti L. On the construction of specifications from requirements[C]//Proc of the 14th Workshop on Requirements Engineering. Rio de Janeiro, Brazil: BDBComp, 2011: 431-442

EXPOSE: Inferring Worst-case Time Complexity by Automatic Empirical Study

Cody Kinnerer [★]

Gregory M. Kapfhammer [★]

Chris Wright [☆]

Phil McMinn [☆]

[★] Allegheny College

[☆] University of Sheffield

Introduction to doubling. A useful understanding of an algorithm’s efficiency, the worst-case time complexity gives an upper bound on how an increase in the size of the input, denoted n , increases the execution time of the algorithm, $f(n)$. This relationship is often expressed in the “big-Oh” notation, where $f(n)$ is $O(g(n))$ means that the time increases by no more than on order of $g(n)$. Since the worst-case complexity of an algorithm is evident when n is large [1], one approach for determining the big-Oh complexity of an algorithm is to conduct a doubling experiment with increasingly bigger input sizes. By measuring the time needed to run the algorithm on inputs of size n and $2n$, the algorithm’s order of growth can be determined [1].

The goal of a doubling experiment is to draw a conclusion regarding the efficiency of the algorithm from the ratio $f(2n)/f(n)$ that represents the factor of change in runtime from inputs of size n and $2n$. For instance, a ratio of 2 would indicate that doubling the input size resulted in the runtime’s doubling, leading to the conclusion that the algorithm under study is $O(n)$ or $O(n \log n)$. Table 1 shows some common time complexities and corresponding ratios.

Ratio $f(2n)/f(n)$	Worst-Case Conclusion
1	constant or logarithmic
2	linear or linearithmic
4	quadratic
8	cubic

Table 1: Conclusions for worst-case time complexity.

Automatic doubling. EXPOSE [2, 3] is a tool to derive an “EXPerimental bigOh” for supporting “Scalability Evaluation”. EXPOSE infers an algorithm’s big-Oh order of growth by conducting a doubling experiment automatically. In order to evaluate an algorithm A , EXPOSE takes as input two functions. The first is a timing function $f(n)$ that runs an implementation of A on the provided input of size n and returns the runtime, and the second is a doubling function $d(n)$ that accepts an input for A and returns an input of size $2n$. After providing EXPOSE an initial input, the tool will output an inferred big-Oh order of growth for A .

EXPOSE derives the worst-case time complexity of A by repeatedly doubling the input until n is large enough that the worst-case time complexity of A is apparent. EXPOSE determines when n is large enough by monitoring the doubling ratio $\frac{f(2n)}{f(n)}$ for multiple iterations of doubling. Using a convergence algorithm, EXPOSE stops the doubling experiment when the doubling ratio reaches a stable value.

To test for convergence, for every time t , where t denotes the number of times the input has been doubled, we record the doubling ratio $r_t = \frac{f(2^t n)}{f(2^{t-1} n)}$. The current ratio r_c is compared to a previous ratio r_p where p is determined by a *lookback* value, such that $p = c - \text{lookback}$. The result of the comparison is a *difference* value, given by $\text{difference} = |r_c - r_p|$. This is then compared to a *tolerance* value, and the experiment is judged to have converged when $\text{difference} < \text{tolerance}$. The *lookback* and *tolerance* values are both configurable parameters.

Early use of the tool revealed that this converge checking rule was not enough, since a very small initial n may complete nearly instantaneously even for multiple rounds of doubling. For example, the time that it takes to sort a list of size 1, 2, 4, 8, . . . , 128 might not even be distinguishable. This would appear to converge to 1, which indicates constant time complexity. To prevent the experiment from incorrectly terminating given a small starting n , EXPOSE requires that a program under study display a ratio of 1 for a *minimum* number of times before judging that the ratio does in fact converge to 1. That is, if $r_c = 1$, $t > \text{minimum}$ must be true, in addition to the tolerance test, before the experiment is declared convergent. The *minimum* value is also a configurable parameter. Because a doubling ratio of 1 signifies constant or logarithmic time complexity, requiring these doubles does not significantly increase the experimentation time needed, all the while providing further assurance that a small ratio is not due to an insufficiently small n .

Implementation. EXPOSE is implemented as a package of classes in the Java programming language [3]. To use EXPOSE to evaluate a new algorithm A , you only need to extend the `DoublingExperiment` class to provide your own f and d functions. The f function should be implemented by providing a `double timedTest()` method, and d should be implemented by providing a `void doubleN()` method. Note that these methods do not accept any parameters, and only `timedTest()` returns a value. The programmer must ensure that `timedTest()` returns the runtime for the current input size, and that when `doubleN()` is called, the input size is doubled. Initializing and storing the input should be handled by the specific implementation. The `runExperiment()` method can be called to conduct a doubling experiment and `printBigOh()` can be called to show the result. Figure 1 shows a complete Java class that conducts a doubling experiment on `QuickSort`; note the simplicity of the implementation when using EXPOSE.

```

public class QuickSortExp extends DoublingExperiment{
    private int size = 10;
    public static void main(String[] args){
        SortingExperiment exp = new SortingExperiment();
        exp.runExperiment(); exp.printBigOh(); }
    protected void doubleN(){ size *= 2; }
    protected double timedTest(){
        int[] n = createInput(size);
        long startTime = System.nanoTime();
        QuickSort.quickSort(n, n.length);
        long endTime = System.nanoTime();
        return (double) endTime - startTime; } }

```

Figure 1: A simple Java class that performs a performance evaluation on the QuickSort algorithm.

Case study: Sorting Algorithms. Included with the EXPOSE tool is an example doubling experiment called *SortingExperiment*. This program provides a number of canonical sorting algorithms with well-known worst-case time complexities. Doubling experiments may be performed on these algorithms by running the command `java SortingExperiment alname`, replacing `alname` with the name of the desired sorting algorithm. Running the command without providing `alname` will show a list of options. When run 1000 times for each of the five provided sorting algorithms, EXPOSE achieves an accuracy of 98.84%.

Case study: *SchemaAnalyst*. In other work [2], we used EXPOSE to perform a comprehensive analysis of the search-based test data generation tool, *SchemaAnalyst*, that generates test suites for relational database schemas [4]. Since it is much more complicated than a sorting algorithm, performing doubling experiments on *SchemaAnalyst* requires more parameters than needed to study sorting. To conduct these experiments, we developed a class called *SchemaExperiment* that extends *DoublingExperiment*. We developed *SchemaExperiment* to allow for conducting doubling experiments using a variety of *SchemaAnalyst* configurations, as well as accessing EXPOSE’s parameters.

```

Usage: <java SchemaExperiment> [options]
Options:
--schema, -s          Select which schema to use
--criterion           Select which criterion to use
--datagenerator       Select which data generator to use
--doubler             Select which schemaDoubler to use
--convergence         Experiment convergent if diff < this
--lookBack           Number of ratios to compare for convergence
--tuningTries         Minimum number of times to doubles before 0(1)
--minDoubles         Minimum number of doubles to try
--giveUp, --maxTime  Max time for a single trial in hours
--help, --usage      Display command line options
-o, --out, --csv     Desired csv filename for saving data
--verbose, --debug   Display verbose output

```

Following the terminology from [5], a doubling experiment to evaluate *SchemaAnalyst* using the AICC criterion, a random data generator, the RiskIt database schema, and the number of NOT NULLs in the schema will run with this command: `java SchemaExperiment --criterion AICC --datagenerator random --schema RiskIt --doubler DoubleNotNullsSemantic`. Although less accurate than in the sorting case study, EXPOSE still successfully revealed meaningful trends in *SchemaAnalyst*’s performance [2].

Deploying on an HPC cluster. Since the performance of *SchemaAnalyst* may depend on a number of factors (i.e., criterion, data generator, schema, and doubling strategy) a comprehensive survey of the parameter space may be conducted by performing a doubling experiment for each configuration. While computationally expensive, an experiment of this scale is possible by using an HPC cluster. Each doubling experiment can be run independently on a separate node of the cluster, and the resulting data can be combined for analysis. Data mining techniques can then be leveraged to interpret an algorithm’s performance trade-offs.

Parameter Tuning. While EXPOSE greatly eases the process of conducting doubling experiments, its accuracy and performance is sensitive to the settings of the system’s parameters. In particular, the *tolerance* and *lookback* values can result in a doubling experiment terminating prematurely or continuing indefinitely. To complicate the issue further, the parameters must be re-tuned based on hardware properties of the machine(s) being used and the performance characteristics of the implementation being studied.

The reliability of the tool and repeatability of its results would be further improved if EXPOSE could select good settings for these parameters automatically. A reasonable parameter tuning strategy could be to run EXPOSE on various algorithms of known worst-case time complexities, such as the sorting algorithms, and lower the *tolerance* threshold until EXPOSE reliably infers the big-Oh time complexities.

Future Work and Conclusion. While we recently used EXPOSE to study search-based test data generation in the domain of relational database schemas [2], the tool is general and can be applied to many other problem domains. Future work includes using EXPOSE to evaluate the efficiency of EVOSUITE’s approach to test data generation for Java programs [6]. In conclusion, EXPOSE makes empirically evaluating the worst-case time complexity of algorithms much more convenient. By automating the process of conducting these experiments, EXPOSE enables large-scale empirical studies that would otherwise be infeasible.

References

- [1] C. C. McGeoch, *A Guide to Experimental Algorithmics*, 2012.
- [2] C. Kinneer, G. M. Kapfhammer, C. J. Wright, and P. McMinn, “Automatically evaluating the efficiency of search-based test data generation for relational database schemas,” in *proc. of 27th SEKE*, 2015.
- [3] C. Kinneer, “EXPOSE software tool,” 2015. [Online]. Available: <https://github.com/kinneerc/ExpOse/>
- [4] G. M. Kapfhammer, P. McMinn, and C. J. Wright, “Search-based testing of relational schema integrity constraints across multiple database management systems,” in *proc. of 6th ICST*, 2013.
- [5] J. Kempka, P. McMinn, and D. Sudholt, “Design and analysis of different alternating variable searches for search-based software testing,” *Theor. Comp. Sci.*, 2015, In Press.
- [6] G. Fraser and A. Arcuri, “1600 faults in 100 projects: Automatically finding faults while achieving high coverage with EVOSUITE,” *Empir. Softw. Engin.*, 2013.

Modeling China Metro Train Route Occlusion Operation Method Based on Time Petri Nets

Ye Zhang¹

Beijing University of Technology¹
Beijing, 100124, China
Ease2003@163.com¹

Yatao Wang², Gang Wang², Jian Sun²

Beijing Jiaotong University²,
Beijing, 100044, China

Abstract—Urban metro signal system is a complex system to guarantee the safe and efficient running of urban metro vehicle, of which any equipment malfunction could induce danger. In the actual operation, the degraded mode of signal system is generally used to organize urban rail transit vehicle when the signal system work abnormally. The traditional degraded mode of urban rail transit signal system is telephone occlusion method, which cannot meet the demand of a large amount of passengers' evacuation for its low efficiency in organizing vehicles. In China, route occlusion, a novel degraded mode, is proposed by a few urban rail transit companies to improve the efficiency of transport in abnormal situation and further ensure rapid evacuation. This paper comparatively analyzes the capability of telephone occlusion and route occlusion to accommodate rail vehicles in an interval between two depots in the same scene. This paper, furthermore, models and analyzes telephone occlusion method and route occlusion method using Petri net in view of its unique advantage to describe asynchronous and concurrent system. Finally, an example of Chang Ping line of Beijing metro is particularized to explain the different efficiency to organize trains in degraded mode of urban rail transit signal system.

Keywords-metro train, route occlusion, telephone occlusion, time petri nets

I. Comparison of Traffic Organization between Route Occlusion and Telephone Occlusion

A. Comparison of Ability to Accommodate Vehicles

Route occlusion and telephone occlusion is the basic alternative occlusion method. Telephone occlusion, whose certificate is the display of signal machine, is a method to confirm the outside interval idle, the station line free and subsequent get access to start into front station through two adjacent station attendants. Route occlusion method need to confirm whether the signal lamp is green or not only, without any telephone communication.

The schematic using route occlusion method and telephone occlusion method to organize trains is shown in figure 1.

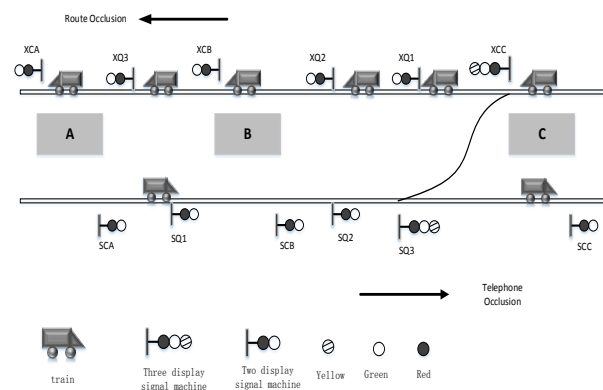


Figure 1 Schematic of route occlusion and telephone occlusion

SCA,SCB,SCC — Respectively represent the ascending signal machine to get out of the station of A,B and C.

SQ1,SQ2,SQ3—Respectively represent the protective signal machine of the demarcation

point of ascending interval.

XCA,XCB,XCC—Respectively represent the descending signal machine to get out of the station of A,B and C.

XQ1,XQ2,XQ3 — Respectively represent the protective signal machine of the demarcation point of descending interval.

Among which SQ3, XCC also play a role of protective signal machine before turnout.

In actual situation, the number of trains with route occlusion method is at least twice as much as telephone occlusion method. And if the number of protective signal machine between two adjacent stations is $n (n > 0)$, $n+1$ trains can be accommodated to travel in theory between the two adjacent stations. However, telephone occlusion method is only able to accommodate one train in any situation. Compared to the phone occlusion method, the advantage of route occlusion will be bigger and bigger along with the length of two adjacent stations.

B. Time Petri Net Modeling

We may get some conclusion from the above analysis that the number of trains that can be accommodated between two adjacent stations with route occlusion method is more than telephone occlusion method. But it is essential for the difference of the number of trains passing through the same station within a certain time between route occlusion method and telephone occlusion method to establish a model of time petri net. The time petri net of route occlusion method and telephone method are as follows:

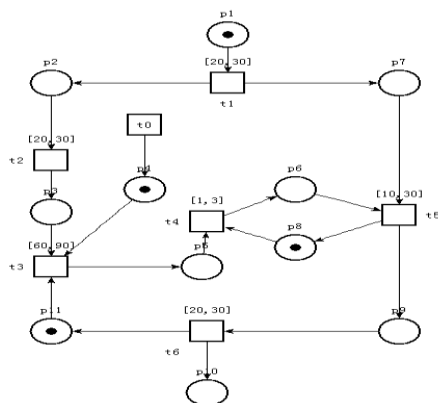


Figure 2 Time petri net of telephone occlusion method

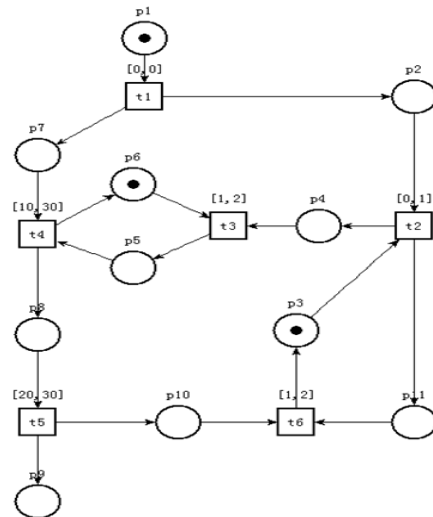


Figure 3 Time petri net of route occlusion method

Table 1 Definitions of some signs of telephone occlusion

place (P)	transition (t)
P1: The train is stopping and waiting for passengers to get on and off.	t1: The attendant of rear station ask to anterior station for starting the train into the interval.
P2: The anterior attendant received ask for starting train and getting access to the station.	t2: The attendant of anterior station check if the line is free.
P3: The anterior station's line is free.	t3: The attendant of anterior station accomplished the check.
P4: The receiving route is ready.	t4: The attendant of rear station light the signal machine.
P5: Mark of allowing train to access.	t5: Closing the door and starting the train.
P6: Symbol of opening signal.	t6: The train get out of the rear station.
P7: The train is waiting for starting.	
P8: Symbol of closing signal.	
P9: Starting the train.	
P10: The train travel into main rail line.	
P11: The interval of two adjacent stations is idle.	

Table 2 Definitions of some signs of route

occlusion

place (P)	transition (t)
P1: The train is stopping and waiting for passengers to get on and off.	t1: The system automatically check interval occlusion.
P2: Section idle.	t2: Locking approach.
P3: Line idle.	t3: Open the signal machine.
P4 : The sign of route locking.	t4: Start the vehicle and enter the route. t5: Close the door and start the vehicle.
P5: The sign of route open.	t6: Unlock the approach.
P6 : The sign of signal closing.	
P7: The train wait to start.	
P8: Starting the train.	
P9: The train travel into the first interval of line.	
P10: The symbol of the train has started;	
P11: A line seizure.	

V. Conclusion

This article analyzed the different features between the two degraded mode of urban rail transit signal system, telephone occlusion method and route occlusion method, discovering that the route occlusion is able to accommodate more trains in one section of two adjacent stations than telephone occlusion method through simulating and depicting the different mode to organize trains. The protective signal machine is constrained by the curve line and climatic, which may conduct accidents that the train operators drive into protective area unexpectedly. Therefore, the position of prospective signal machine should be placed reasonably, for instance, increasing the repeating signal.

ACKNOWLEDGMENT

This work was supported by the Beijing Postdoc Program (2014ZZ-64).

References

- [1] A. Sahraoui, H. Atabakhche, M. Courvoisier, and R. Valette, Joining Petri nets and knowledge based systems for monitoring purposes[C]. in Proc. IEEE Robotics and Automat. Conf., Raleigh, NC,1987,1861-1867.
- [2] M. C. Zhou, F. DiCesare, and A. A. Desrochers, A top-down modular approach to synthesis of Petri net models for manufacturing systems[C] in Proc. 1989 IEEE Robotics and Automat. Conf., Scottsdale, AZ, May 1989, 534-539.
- [3] Huifang Li, Renhou Li. Studies about Schedulability Analysis of Timing Constraint Petri Nets [J]. Computer Science, 2000, 27(3).
- [4] Antonio Cerone, Andrea Maggiolo-Schettini. Tutorial Time-based expressivity of time Petri nets for system specification [J]. Theoretical computer Science, 1999(216).
- [5] Woei-Tzy Jong, Yuh-Shin Shiau, Yih-Jen Horng, Hsin-Horng Chen, Shyi-Ming Chen.Temporal Knowledge Representation and Reasoning Techniques Using Time Petri Nets[J].IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics, 1999, 29(4).
- [6] Wei Sheng-jun, Hu Chang-zhen, Sun Ming-qian. Method of Dynamic Knowledge Representation and Learning Based on Fuzzy Petri Nets [J].Journal of Beijing Institute of Technology, 2008, 17(1).
- [7] Jia Lixin, Xu Junyi, Ru Feng. Fuzzy Petri Net Based Formalized Reasoning Algorithm with Applications[J].Journal of Xi'An Jiao Tong University, 2003, 37(12).
- [8] Song Yu-bo, Jiang Zhao-yuan, Mu Hai-bo. Algorithm for Seeking the Minimal Cost and Maximal Flow Based on Petri Net[J].Journal of Lanzhou Jiaotong University, 2011, 30(3).
- [9] Li Peng, Li Huimei. Research on Efficiency and Safety of Urban rail Transit with downgrade[J].China Science & Technology Panorama Magazine, 2011(9).
- [10] Li Yuhui. Application of Telephone Occlusion Method in Train Operation Organization of Metro [J].Modern Urban Transit, 2010(2).
- [11] Zhang Yonggai, Zeng Haijun.Call blocking method in Metro downgraded security risks in operation and Countermeasures[J]. Shangpin yu Zhiliang, 2013(1).
- [12] Zhou Mengxiang. City track traffic organization method based on telephone occlusion [J].Technology And Life, 2011(5).
- [13] Jiang Zhibin, Ji Tingting. URT Train Regulation Strategy Based on Passenger Flow Influence[J].Urban Mass Transit, 2014, 17(1).
- [14] Yan Dong. Discussion on the premise of improving telephone occlusion to ensure the safety of driving efficiency [J]. Chengshi Jianshe Lilun Yanjiu, 2012(7).
- [15] Xie Bin, Zhou Rui, Laiqi. Optimization of city rail traffic telephone occlusion method[J].transportation Enterprise Management, 2012, 27(4).
- [16] Merlin P.M., A Study of the Recoverability of Computing Systems. Irvine: Univ. California, PhD Thesis, 1974. Available from ANN Arbor: Univ. Microfilms, No.75-11026.
- [17] ZHANG Xiao-hui, HE Jie, SUN Jing , ZHANG Di. The reliability analysis of the subway signal system based on Petri Nets [J].SHANDONG JIAOTONG KEJI, 2009(4).
- [18] ZHONG WEN YAN. Modeling and Analysis for Railway Signaling System with Petri Nets. [D].Southwest Jiaotong University , 2005.
- [19] Fu Jiehui. Research on Fault Diagnosis Methods Based on Petri Net [D].Southeast University, 2004.

System Architecture of a Train Sensor Network for Ubiquitous Safety Monitoring

Guoqiang Cai¹, Limin Jia¹, Ji'an Sun¹,

Kun Zhang¹, Shuai Feng¹, Mingming Zheng¹

State Key Laboratory of Rail Traffic Control and Safety,
Beijing Jiaotong University, Beijing, 100044, China¹

Guoqiangcai@163.com

MengChu Zhou²

Department of Electrical and Computer Engineering, New
Jersey Institute of Technology, Newark, NJ 07102, USA²

ABSTRACT—Train safety monitoring and fault diagnosis are critically important because of the disastrous results caused by train collisions and derailments. Train safety protection sensors network is capable of autonomously monitoring the working condition and actively control faults. A number of strategically placed sensors in the vehicles form a network that can monitor various vital parameters and provide real-time prompt to train driver and dispatcher. Designing such networks faces a number of challenging tasks, as one needs to address some conflicting requirements for quick diagnosis, collaborative decision making, achieving high precision and reliability. This paper presents an on-line Train Safety Sensor Network (TSSN) architecture, discusses its hardware and software structure for ambulatory failure status monitoring. The network consists of multiple sensor layers that monitor train's electrical and mechanical activities, a train data center and a ground data analysis server. The server implements fault diagnosis based on a Fault Tree Analysis method (FTA). The results shows that the sensor network contributes to higher train safety guarantee.

Keywords-Train safety, Sensor network, FTA

I. Introduction

Real-time train safety monitoring is a key technology in helping proactive and affordable train healthcare. It allows workers to continuously monitor changes in vital signs and provide feedback to improve maintenance schedule. Recent technological advances in sensor networks enabled the design and proliferation of wireless sensor networks capable of autonomously exerting early control-related parameters under safety thresholds, preventing some otherwise safe cases from “developing” into dangerous ones. The systems can early alert maintenance personnel with a diagnostic procedure via a friendly user interface and optimal supervised recovery adhering to repair standards and guidelines. Critical development in a train sensor network is data acquisition, wireless communication and vehicle reliability analysis. However, train faults resulting in fatal accidents frequently happen in the entire world. Real-time train working condition monitoring and systematical analysis are challenging in a transportation domain.

During the last few years there has been a significant increase in the number and variety of train monitoring

devices and systems. However, their acceptance is limited due to the following important restrictions. Traditionally, they are used to collect data only. Data processing and analysis are performed offline, making them impractical for continual monitoring and early detection of degraded components in a train. In addition, their individual sensors often operate as stand-alone systems and usually do not offer flexibility and integration with a networks. At present, train safety network have some applications to multifunction vehicle buses[4]. A navigation and communication monitoring system of space-earth integration for railway safety based on Chinese Area Positioning System (CAPS), can locate a train and communicate its positioning data to the monitoring center [2]. Accidents may stem from decisions and actions taken at times and places quite distant from their final location [1,3,5].

This paper proposes a TSSN architecture. It implements a train-ground wireless on-line tele-monitoring system. It consists of in-train physiological sensors that monitor train health, in-train datacenter and ground server. We describe the hardware and software organization of TSSN. The result of an example shows that the TSSN is practical and meaningful for qualitative and quantitative reliability analysis and useful in practice.

II. System Architecture

A train is a sophisticated mechanical-electrical system many researchers have devoted their efforts to its safety monitoring. It consists of several vehicles. Vehicle faults are the biggest accident source in rail transportation. For example, ill-function of its door system can lead to sudden train braking, thereby resulting derail accident.

Our work intends to conduct train safety monitoring. The monitoring content is composed of the running condition of vehicles and vehicle loading and infrastructure conditions. Each vehicle itself records its working states and stores in vehicle diagnosis node according to a time sequence. Several train data centers send data to Ground Data Center (GDC) by wireless communication channel. GDC synthesizes all of monitoring information based on certain criteria and

decision making methods, provides timely diagnosis results to avoid an accident and predicts hidden dangers. The TSSN architecture and decision making process are

show in Figure 1.

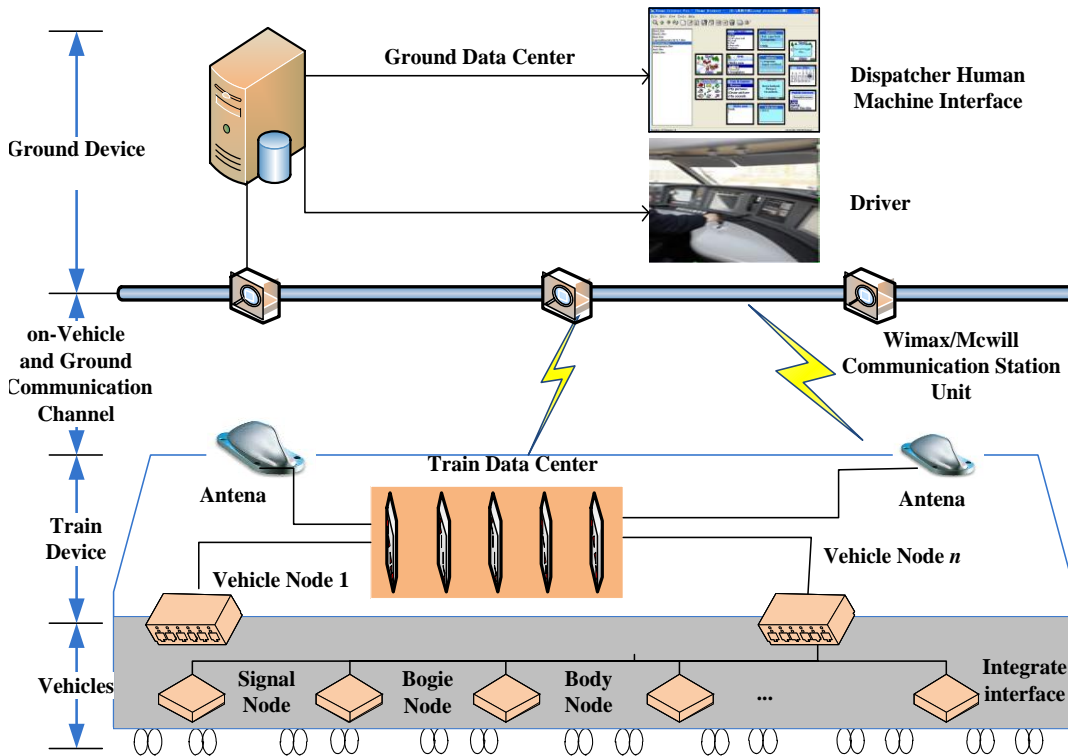


Figure 1. Proposed system architecture

The key technology of TSSN uses distributed sensors to collect online data. TSSN integrates vehicles' information addressing different aspects of inter-vehicle connection, builds an available network structure, a circulation mechanism, a diagnosis data fusion interface.

III. System Reliability Analysis of TSSN

The most important function of TSSN is reliability analysis. For critical components, failure rates of events are calculated via FTA models.

A. FTA Model and State Transfer Algorithm

The events in a fault tree model include basic events, intermediate events and top event. All the events have only two states: failure or normal. The corresponding probabilities are a failure rate and a success rate whose corresponding state values are expressed by 0 and 1.

The codes according with the basic events are in Table 1, the Instantaneous Failure Rate data are collected from Shanghai Metro historical maintain records.

Table 1. Failure modes and their characteristics

Code	Name	Instantaneous Failure Rate
T	Door System Open Failure	0.002707089
X ₁	Speed Sensor 1	0.001152
X ₂	Speed Sensor 2	0.001152
X ₃	Speed Sensor 3	0.001152
X ₄	Speed Sensor 4	0.001152
X ₅	Open / Close Button	0.0005
X ₆	EDCU Failure	0.000813

X ₇	Open / Close Solenoid Valve	0.000356
X ₈	Emergency Unlocking Handle	0.00125
X ₉	Unlock Solenoid Valve	0.000749
X ₁₀	Unlock Signal	0.000962
X ₁₁	Screw / Nut Failure	0.0001
X ₁₂	Rail Choked	0.00022
X ₁₃	Rubber Pad Damage	0.0004
X ₁₄	Motor Failure	0.000058
X ₁₅	Coupling Failure	0.00025
W ₁	Speed Sensor Failure	7.944E-06
W ₂	Unlock Failure	9.362E-07
W ₃	Auto Unlock Failure	1.203E-06

B. Statistical Simulation

The simulation process is implemented in C + +. After 10,000 simulation runs, a series of simulation data

are obtained. Then the processing results are displayed in Figure2.

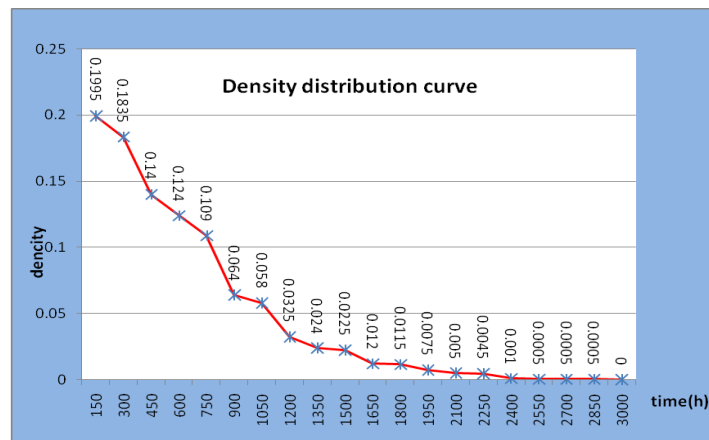


Figure 2. Density curve of failure distribution

IV. Conclusions

In this paper we describe both hardware and software architecture of TSSN. The architecture leverages off-the-shelf commodity computing platforms. The reliability of application and simulation software deployed upon real-time operating system for embedded sensor networks. TSSN promise a ubiquitous monitor vital working condition parameters. It provides a shift from passive failure management toward more proactive preventive accident care and reduces failure occurrence frequency.

Acknowledgment

This work was supported by the National Science Foundation of China under grant number 863 Rail Safety Early-warning and Guarantee Technologies Program (2012AA112403)(2011AA110501).

References

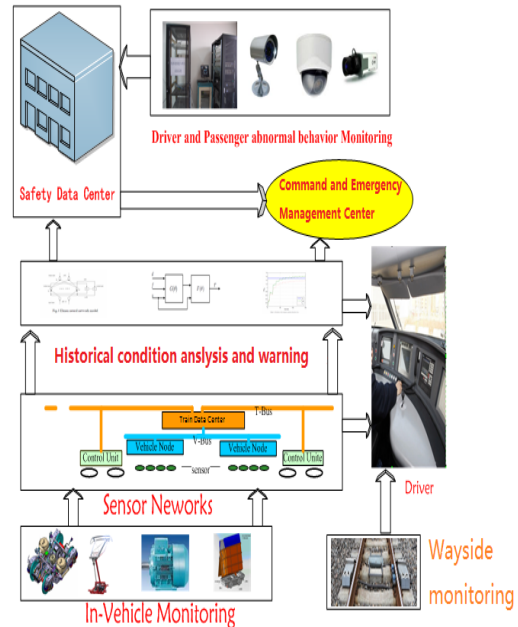
- [1] Hale, A.R. Railway Safety Management: the challenge of the new millennium, *Safety Science Monitor*. 4(1), pp. 1-15, 2000.
- [2] Hu, Z.Q., Cui, J.X., Zhang, J., Zhang, L.R, and Lv, Ch. Railway Safety Monitoring System Based on CAPS. *Proceedings of 2013 Fifth International Conference on Measuring Technology and Mechatronics Automation*. pp. 838-40, HongKong, Jan 16-17 2013.
- [3] Reason, J. Managing the Risks of Organizational Accidents. *Ashgate*.1997.
- [4] Sun, Y. and Li, X. Design of safety monitoring system of locomotive network based on multifunction vehicle bus, *Railway Computer Application*, 16(4), pp. 20-24, Apr. 20 2007.
- [5] Turner, B.A. and Pidgeon, N.F. Man-made disasters: why technology and organizations (sometimes) fail. *Safety Science* 34 pp. 15-30, 2000.

System Architecture of a Train Sensor Network for Ubiquitous Safety Monitoring

Guoqiang Cai¹, Limin Jia¹, MengChu Zhou², Ji'an Sun¹, Kun Zhang¹, Shuai Feng¹, Mingming Zheng¹

Train safety monitoring and fault diagnosis are critically important because of the disastrous results caused by train collisions and derailments. Train safety protection sensors network is capable of autonomously monitoring the working condition and actively control faults. Real-time train safety monitoring is a key technology in helping proactive and affordable train healthcare. It allows workers to continuously monitor changes in vital signs and provide feedback to improve maintenance schedule. Recent technological advances in sensor networks enabled the design and proliferation of wireless sensor networks capable of autonomously exerting early control-related parameters under safety thresholds, preventing some otherwise safe cases from “developing” into dangerous ones.

Our work presents an on-line Train Safety Sensor Network (TSSN) architecture, discusses its hardware and software structure for ambulatory failure status monitoring.



TSSN can early alert maintenance personnel with a diagnostic procedure via a friendly user interface and optimal supervised recovery adhering to repair standards and guidelines. TSSN promise a ubiquitous monitor vital working condition parameters. It provides a shift from passive failure management toward more proactive preventive accident care and reduces failure occurrence frequency. FTA model and simulation analysis show that TSSN contributes to train system reliability analysis.

Agile Practices in Maturity Model for Testing: an Experience Report

Ana Paula C. C. Furtado, Suzana Sampaio^{1,2}
 Informatics Center - CIn
¹Federal University of Pernambuco
 Recife, PE, Brazil
 {apccf, scbs2}@cin.ufpe.br

Ermeson Andrade
²Federal Rural University of Pernambuco
 Recife, PE, Brazil
 ermeson@deinfo.ufrpe.br

Ivaldir de Farias Junior, Marcos Wanderley
³SOFTEXRECIFE
 Recife, PE, Brazil
 {junior, marcos}@recife.softex.br

Abstract—Software testing is an important tool for ensuring that software products produced and launched on the market reach the appropriate quality standards. Testing maturity models such as MPT.BR or TMML, appear in the software scenario as a way to support introducing elements that are essential for developing the discipline of software testing within organizations. Together with this reality, it is observed that there is a strong trend towards using agile methods in software development. Therefore, this paper presents an experience report on the use of agile practices together with MPT.BR.

Keywords— software testing, maturity models, agile practices.

I. INTRODUCTION

In today's context of software development, given the increase in the demand for products and the reduction in the number of qualified personnel to develop them, quality is a key concept in strategies for winning a share of this market. In the broad context of software quality, one of the common characteristics observed is that it is essential for the specification given to be adequate and in accordance with clients' needs. Hence, testing software before delivering products to clients is one of the ways to achieve quality.

In this context, maturity models under test, such as MPT [1], TMM [2] and TMML [3], are guides that assist organizations to introduce the essential elements for the development of the discipline of testing, given that it is not always known where to begin to define a testing process. Agile methods, for their part, appear in the software setting as an alternative to software development, which is faster and more readily adaptable to the client's needs. The practices arising from this context are also instantiable for testing processes, which should be interpreted as if one can map the concepts of agility for testing activities in the software development scenario.

Therefore the aim of this paper is to present an experience report on how some process areas of MPT.BR were implemented together with agile methods based on data collected from implementing the model in 27 software engineering organizations over the last 4 years.

II. EXPERIENCE REPORT

This Section describes an experience report based on the experience of implementing MPT.BR in various organizations all over Brazil, in which agile practices were adopted in the testing environment. Table I shows the mapping of the process areas in agile implementations of MPT.BR.

TABLE I. AGILE IMPLEMENTATION IN MPT.BR

Process Area	Practices	Agile Implementation
GPT	GPT4	Scrum Taskboard, Kanban Board, Sprint Backlog, Improvement Backlog
	GPT5	Planning Poker, Ideal Days, Relative Sizing
	GPT6	Short Iterations, Sprints
	GPT9	Daily Meetings to Identify Risks
	GPT13	Agile Metrics, such as Sprint Burndown Chart
	GPT16, GPT18	Daily Meetings, Sprint Review, Retrospective
	GPT17	Continuous Feedback, Client Collaboration
	GPT19, GPT20	Daily Meetings
PET	PET1, PET2	Test Driven Development – TDD Behavior Driven Development - BDD
GRT	GRT1	User Stories, Backlog Item
FDT	FDT3	Sprint Review Restrospective
GDQ	GDQ2	Daily Meetings
MAT	MAT1	Agile Metrics, such as Sprint Burndown Chart
	MAT4	Daily Meetings, Restrospective
TES	TES3	Pair Programming Peer Review
	TES4	Daily Meetings
GDD	GDD1	Daily Meetings, Retrospective
AET	AET1 AET2 AET3	Test Driven Development – TDD Behavior Driven Development - BDD

The details of how agile practices were mapped to the process areas can be observed as described below:

- **GPT**: agile practices were used to introduce the concept of test sprints, in which the requirements to be tested are arranged in backlog and made visually available using scrum boards (or kanban boards). moreover, planning poker, relative sizing and ideal days can be used as techniques for estimating the size of the stories to be tested, particularly as a metric so as to construct sprint burndown, team velocity, lead time, etc. Daily meetings were introduced to monitor the project and as a mechanism to identify and stay abreast of project risks.
- **PET**: test driven development (TDD) or behavior-driven development (BDD) are techniques that could be used to identify the project's test cases and satisfy the demand of the process area.
- **GRT**: instead of formal requirements, the scope of projects could be organized using user stories that are part of the project backlog.
- **FDT**: during the sprint review, the tested items can be packaged so they can be delivered and the test environment can be clean, thus satisfying part of what the test closure process area requires. In addition, the practice of retrospective adds on the lessons learned, thus bringing an implemented agile practice to FDT.
- **GDQ**: for this process area, the practice of daily meetings can be implemented to include the reporting of items on the quality of the project.
- **MAT**: the agile metrics suggested by scrum, such as burndown, velocity, and lead time can be used as an option for the indicators of the test project. In addition, the daily meetings and retrospectives can be used to report on these results.
- **TES**: the static test can be conducted by pair programming, where not only the revision of the code developed is observed but also the dissemination of knowledge. In addition, the daily meetings can be used as a moment to analyze the data from the reviews and to standardize communication with the team.
- **GDD**: the daily meetings and retrospectives can also be used to identify the root causes of the defects found.
- **AET**: the test can be automated based on the BDD and TDD techniques, besides which automating the test itself is already considered the introduction of agile design practices.

A. MPT.BR AGILE CERTIFIED ORGANIZATIONS

Based on the results of the current implementations, the consolidated situation is that, of a total of 16 existing process areas in MPT.BR, so far 10 have been implemented using agile methodologies, i.e. 63% of the model has already been instantiated in an agile way. These data were obtained by

analyzing the implementation of MPT.BR in the 27 companies that have been evaluated between 2010 and 2014.

The number of implementations with agile methodologies (16) is greater than the number of companies that have implemented it using the traditional method (11), which is another indication that the current trend of software development is to use agile methodologies. The largest number of evaluations so far is at the first level of maturity and 75% of companies have also used agile methods.

III. CONCLUSION AND FUTURE RESEARCH STUDIES

This article presented a theoretical framework for agile methodologies and how these methodologies were instantiated in conjunction with a testing maturity model.

The use of agile methods has been observed as a trend in the area of software development, and this can also be observed when the aspect of software development is directly related to the testing processes of a given organization.

From the data collected from the assessments of the MPT.BR from January 2010 until April 2014, it was observed that most of the companies evaluated made use of agile methodologies when instantiating their processes. It was also observed that the use of agile methods in conjunction with the testing maturity model is not restricted to how large or small an organization is because it was conceived in small, medium and large companies.

Therefore, based on the results obtained so far, it is predicted that this study can be complemented by the following future studies:

- Seeking ways to conduct Non-Functional Testing of agile ways to support the implementation of the TNF - Non Functional Testing process area;
- Seeking agile practices to carry out the process area of CEP - Statistical Control of Processes;
- Enhancing the automation of the tests over all process areas and maturity levels of MPT.BR in order to maximize the results obtained; and
- Understanding the reasons that lead an organization to choosing either an implementation with a traditional approach or an agile approach.

References

- [1] A. Furtado, M. Gomes, E. Andrade, I. de Farias Junior (2012). MPT.BR: A Brazilian Maturity Model for Testing Published in: Quality Software (QSIC).
- [2] E. Veenendaal and R. Swinkels (2002) Guidelines for Testing Maturity. Available at <http://goo.gl/0n5d3V> captured 26/04/2014.
- [3] E. Veenendaal(2012). Test Maturity Model Integration Release 1.0. TMMi Foundation, Ireland. Available at www.tmmifoundation.org/downloads/TMMi/TMMi%20Framework.pdf captured 26/04/2014.

Authors' Index

Abou Khaled, Omar	445
Abran, Alain	158
Abrantes, Joilson	359
Abreu, Fernando	279
Adam, Sebastien	158
Adjoyan, Seza	225
Adornes, Daniel	603
Affonso, Frank José	24
Aguiar, Rui	195, 353
Akram, M.Usman	260
Aktouf, Oum-El-Kheir	74
Alencar, Lucas Andre	238
Alencar, Paulo	644
Almeida, Eduardo	680
Almeida, Hyggo	122
Alshaikh, Zeyad	676
Alvares, Luis Otavio	238
Ando, Reou	524
Andrade, Aline	542, 609
Andrade, Ermeson	717
Anvik, John	435
Aparecida de Almeida Ribeiro, Angélica	449
Araujo Ramos, Felipe Barbosa	99
Araujo Soares, Gustavo	99
Ariss, Omar	405
Assunção, Joaquim	565
Ayaz, Sadaf	260
Bagheri, Ebrahim	439
Barbosa Araújo Ramos, Felipe	93
Barbosa, Simone	13
Barcellos, Monalessa P.	500
Barchet, Catherine	303
Behl, Sanjiv	202
Ben Abdallah, Hanêne	654
Ben Saber, Haifa	172
Bertuol, Gelson	217
Bezerra, Byron L. D.	689
Bhalerao, Deepti	110
Binder, Walter	579
Bogorny, Vania	238
Bonifacio, Rodrigo	597
Borisenko, Konstantin	699
Bouassida, Nadia	654

Brito, Maria	176
Brondani, Camila	217
Bruegge, Bernd	146
Burnay, Corentin	325
Cagnin, Maria Istela	221
Cai, Guoqiang	713, 716
Cai, Xuyang	375
Cao, Bei	299
Carneiro, Glauco	279
Carranza Chávez, Bonnie G.	636
Castro, Thiago M.	597
Cavalcante, Emanuelle	479
Cavalcanti Furtado, Ana Paula	717
Cha, Sungdeok	81
Chang, Ling-Hua	202
Chang, Shi-Kuo	51, 57, 58, 273, 551
Che, Meiru	152
Chen, Deng	347, 455, 591
Chen, Haiming	207
Chen, Lin	128, 254, 261
Chen, Liqiong	569
Chen, Mei	87
Chen, Mingming	381
Chen, Wen-Hui	51
Chen, Xiangping	461
Chen, Yuting	315
Chen, Zhifei	128
Coelho, Roberta	359
Cois, Constantine Aaron	700
Colace, Francesco	58
Confort, Valdemar	506
Conte, Tayana	1, 13, 134, 479, 485, 670
Cowan, Donald	644
Cui, Li	207
Cunha, Francisco J. P.	644
de Paiva Oliveira, Alcione	449
Dias, Sergio M.	250
Ding, Junhua	573
Ding, Wei	387
Domingues, Anderson	211
Du, Miao	512
Du, Yuheng	411
Duran, Adolfo	293
Durelli, Vinicius	335
El Boussaidi, Ghizlane	158

El-Kharboutly, Rehab	231
Eler, Marcelo Medeiros	335
Elloumi, Mourad	172
Endo, André Takeshi	335
Fagerholm, Fabian	393
Fakhfakh Akrouf, Sana	704
Falbo, Ricardo De A.	500
Fan, Guisheng	104, 569
Fani, Hossein	439
Far, Behrouz	182
Farias, Kleinner	530, 640
Faulkner, Stéphane	325
Fernandes, Luiz Gustavo	603
Fernandes, Paulo	565
Fernandes, Ricardo	597
Ferreira, Bruna	485
Fiondella, Lance	189
Fonseca, Vinícius S.	500
Fontoura, Lisandra	217, 303
Fontán, Carina	42
Francisco Da Matta Vegi, Lucas	449
Freitas, Elyda L. S. X.	689
Fukazawa, Yoshiaki	524, 615
Gao, Jerry	74
Gao, Kehan	423
García-Castro, Raúl	630
Gokhale, Swapna	189, 231
Gomede, Everton	309
Goncales, Lucian	640
Gonçales, Lucian	530
Gonçalves, Rafael	36
Goswami, Anurag	664
Greco, Luca	58
Gresse von Wangenheim, Christiane	36
Griebler, Dalvan	603
Gu, Junzhong	162
Guimarães, Everton T.	644
Gärtner, Stefan	399
Gómez-Pérez, Asunción	630
Hallstrom, Jason	411
Han, Ah-Rim	81
Hanai, Yoshiiku	524
Hazimeh, Hussein	445
He, Jia	66
He, Sen	676

He, Xudong	559
Heinrich, Robert	399
Hesse, Tom-Michael	146, 399
Hirayama, Hideaki	701
Hori, Akihiro	623
Horita, Hiroki	701
Hu, Jianpeng	299
Hu, Qingcheng	244
Huang, Chao	364
Huang, Yuan	461
Huchard, Marianne	658
Hughes, Shiree	411
Inoue, Sakae	524
Jarkass, Iman	445
Jenkins, Marcelo	429
Jeong, Sehun	81
Jia, Limin	512
Jiang, Shujuan	347
José de S. Fonseca, Emilio	449
José, Bernardo	670
Ju, Jianping	591
Junior, Ivaldir	717
Jureta, Ivan	325
Jürjens, Jan	399
Kalinowski, Marcos	1
Kanazawa, Masanobu	524
Kapfhammer, Gregory	341, 709
Kaur, Arshinder	650
Kazman, Rick	700
Kchaou, Dhikra	654
Kholod, Ivan	699
Khoshgoftaar, Taghi	369, 423
Kinneer, Cody	341, 709
Krishna, Aneesh	650
Kuehlwein, Arthur	146
Kulesza, Uirá	597
Laboon, Bill	551
Lacher, Lisa	393
Landre, Geraldo	221
Laser, Marcelo	211
Ledur, Cleverson	603
Lee, Hyo-Cheol	30
Lee, Seok-Won	30, 168
Leite, Gustavo	24

Lemma, Saverio	58
Lessa, Ivan	279
Li, Dong	207
Li, Xun	455
Li, Zhi	706
Liang, Peng	7, 387
Lima, Anderson	122
Lin, Jingjian	555
Lin, Lan	66
Lin, Wen-Chyi	51
Linpeng, Huang	299
Lisboa-Filho, Jugurta	449
Liu, Guoyuan	706
Liu, Su	559
Liu, Yan	87, 283, 331
Liu, Zunhe	331
Lombardi, Marco	58
Longo, Douglas Hiura	319
Lopes, Adriana	13
Lopes, Lucelene	565
Lownes, Nick	189
Lucena, Carlos J. P.	644
Luna, Alexandre	46
Luo, Ruici	116
M Subramanian, Chitra	650
Ma, Hui	512
Ma, Wanwangying	128
Ma, Yutao	381
Macedo Rodrigues, Elder	211
Maciel, Alexandre M. A.	689
Maciel, Rita Suzana Pitangueira	542, 609
Magalhaes, Ana Patricia	542, 609
MAHDI, Walid	704
Maia, Marcelo	368
Maldonado, José Carlos	134
Malucelli, Andreia	267, 490
Mani, Nariman	536
Maria Maciel Braga Villela, Regina	449
Marinho, Marcelo	46
Marques, Anna Beatriz	13
Martoglia, Riccardo	140
Matturro, Gerardo	42
McMinn, Phil	341, 709
Melgar, Andrés	636
Menolli, Andre	267, 490
Miranda de Barros, Rodolfo	309
Mokni, Abderrahman	658

Monteiro, Miguel	279
Mostafa, Shaikh	676
Moura Costa, Antonio Alexandre	93, 99
Moura, Hermano	46
Mugellini, Elena	445
Murillo-Morera, Juan	429
Méndez Fernández, Daniel	1
Münch, Jürgen	393
Nakagawa, Elisa Yumi	24, 293
Nakagawa, Hiroyuki	473
Nakstad, Frederik	615
Namba, Katsushi	524
Napolitano, Amri	369, 423
Neto, Sebastiao M.	250
Noureddine, Hassan	445
Nygard, Kendall	393
Ohsuga, Akihiko	701
Oinuma, Morihide	623
Oliveira de Almeida, Hyggo	93, 99
Oliveira, Brauner R. N.	293
Oliveira, Diógenes R. F.	689
Oliveira, Edson	485
Oliveira, Flavio	211
Oliveira, Rafael A. P.	24
Oliveira, Toacy	530, 640
Ono, Hiroyuki	524
Ouyang, Zhaofeng	706
P Gopalan, Raj	650
Paech, Barbara	146, 399
Pagels, Max	393
Paiva, Debora	221
Palmeira, Alisson	597
Paraiso, Emerson	467
Park, Soojin	18
Park, Young B.	18
Pei, Xin	104
Peng, Li	455
Pereira, Oscar	195, 353
Perkusich, Angelo	93, 99, 122
Perry, Dewayne	152
Petriu, Dorina	536
Prikladnicki, Rafael	1, 670, 693
Qi, Zhengwei	364, 579
Qin, Yong	512

Qu, Binbin	591
Radulovic, Filip	630
Raffaeta, Alessandra	238
Rahme, Jean	287
Raschetti, Florencia	42
Regateiro, Diogo	353
Reinehr, Sheila	267, 490
Renso, Chiara	238
Reussner, Ralf	399
Rivero Cabrejos, Luis Jorge Enrique	134
Rivero, Luis	479
Roehm, Tobias	146
Rolim de Sousa, Reudismam	99
Rolim, Reudismam	93
Ruhroth, Thomas	399
Sadjadi, S. Masoud	684
Sahar, Sadaf	260
Salehian, Soheil	182
Sampaio, Suzana	46, 717
Santoro, Flavia	506
Santos, Daniel Soares	293
Santos, Gleison	506
Santos, Marcos	176
Saputri, Theresia Ratih Dewi	168
Sato, Seiji	524
Schneider, Kurt	399
Scholl, Murillo	530
Scholl, Murilo	640
Schreiber, Daniel	267
Seriai, Abdelhak	225
Shen, Beijun	315, 375
Shen, Ning	417
Shorov, Andrey	699
Silva, Ivonei	680
Silva, Luís Alvaro	303
Silva, Williamson	485
Simões, David	195
Singh, Abhinav	664
Soares, Gustavo	93
Sone, Kazutaka	524
Song, Kwangsik	81
Song, Mark A. J.	250
Souza, Iuri	680
Souza, Paulo	176
Souza, Simone	176
Spinola, Rodrigo	1

Studený, Angelika	565
Su, Chao	585
Sun, Haiyang	364
Sun, Hao	520, 585
Tahara, Yasuyuki	701
Taheri, Mohsen	684
Takada, Shingo	623
Takahashi, Hitoshi	473
Tang, Antony	387
Tangari, Guilherme	368
Tanno, Haruto	623
Tasse, Josee	496
Terenciani, Marcelo	221
Thiago Da Silva, Rafael	309
Thomas, Christopher Lee	51
Tironi, Huander	490
TMAR, Mohamed	704
Trinkenreich, Bianca	506
Tsuchiya, Tatsuhiro	473
Tunnell, James	435
Uchida, Chihiro	524
Uehara, Tadahiro	74
Urtado, Christelle	658
Vacari, Isaque	693
Vale, Tassio	680
Valentim, Natasha	670
Van Vliet, Hans	387
Vauttier, Sylvain	658
Veronez, Mauricio	530
Veronez, Maurício	640
Viana, Marx L.	644
Vilain, Patrícia	319
Vilar, Rodrigo A.	122
Vincent, Jean-Marc	565
Vincenzi, Auri Marcelo Rizzo	134
Wagner, Stefan	1
Walia, Gursimran	393, 664
Wanderley Gomes, Marcos André	717
Wanderley, Gregory	467
Wang, Beibei	128
Wang, Haofen	375
Wang, Huanjing	369
Wang, Jingtao	273
Wang, Rongcun	347, 455, 591

Wang, Shuohong	364
Wang, Xiaoyin	676
Wang, Yatao	712
Wang, Yongming	162
Wang, Yue	520, 585
Wang, Ziyuan	546, 705
Washizaki, Hironori	524, 615
Wei, Wei	455, 591
Woodside, Murray	536
Wright, Chris	341, 709
Wu, Di	254, 261
Xiang, Chengcheng	579
Xie, Kaibin	207
Xing, Chunxiao	244
Xu, Baowen	128, 254, 261
Xu, Dianxiang	417
Xu, Haiping	110, 287
Xu, Weifeng	405
Xu, Xinhui	244
Xu, Yangyang	87, 283
Xuan, Jifeng	555
Xue, Yufeng	66
Yamamoto, Mikihiko	524
Yan, Jun	555
Yang, Hui	7
Ye, Wei	116
Yu, Huiqun	104, 569
Yung, Duncan	58, 551
Zarate, Luiz E.	250
Zegarra, Emilio	273
Zeng, Qingkai	520, 585
Zhang, Dongmei	573
Zhang, Huaxi Yulin	658
Zhang, Shikun	116
Zhang, Tao	74
Zhang, Yanduo	455, 591
Zhang, Ye	711, 712
Zhang, Yong	244
Zhang, Yunpeng	417
Zhao, Shixiong	315
Zheng, Jiabin	283
Zhong, Hao	315
Zhou, Mengchu	713
Zhou, Yuming	254, 261
Zhu, Jiangang	375

SEKE2015

Authors' Index

Zorzo, Avelino F.
Zou, Qiwen

211
461

Program Committee Reviewers' Index

Silvia T. Acuña	Universidad Autonoma de Madrid
Shadi Alawneh	Faculty of Engineering & Applied Science, Memorial University
Taiseera Albalushi	
Mark Allison	University of Michigan-Flint
Izzat Alsmadi	Boise State University
John Anvik	Department of Computer Science, Central Washington University
Doo-Hwan Bae	KAIST
Ebrahim Bagheri	Ryerson University
Hamid Bagheri	George Mason University
Xiaoying Bai	
Purushotham Bangalore	University of Alabama at Birmingham
Ellen Barbosa	ICMC/USP
Fevzi Belli	Univ. Paderborn
Ateet Bhalla	Oriental Institute of Science and Technology, Bhopal, India
Swapan Bhattacharya	
Alessandro Bianchi	Department of Informatics - University of Bari
Ivo Bukovsky	Department of Instrumentation and Control Engineering, Faculty of Mechanical Engineering, Czech Technical University in Prague
Keith C.C. Chan	The Hong Kong Polytechnic University
Chih-Hung Chang	Department of Information Management, Hsiuping University of Science and Technology
Kuang-Nan Chang	
Shi-Kuo Chang	University of Pittsburgh
Meiru Che	The University of Texas at Austin
Shu-Ching Chen	Florida International University
Wen-Hui Chen	Taipei University of Technology
Zhenyu Chen	Nanjing University
Stelvio Cimato	Dipartimento di Informatica, Università degli Studi di Milano
Nelly Condori-Fernández	VU University of Amsterdam
Fabio Costa	Federal University of Goias
Maria Francesca Costabile	Dipartimento di Informatica - University of Bari
Jose Luis de La Vara	Simula Research Laboratory
Massimiliano Di Penta	Dept. of Engineering - University of Sannio, Italy
Scott Dick	University of Alberta
Junhua Ding	East Carolina University
Fei Dong	
Derek Doran	Wright State University
Weichang Du	University of New Brunswick
Philippe Dugerdil	HEG-Univ. of Applied Sciences, Department of Information Systems
Christof Ebert	Vector
Ali Ebneenasir	Michigan Technological University
Raimund Ege	Northern Illinois University

Magdalini Eirinaki	Computer Engineering Dept, San Jose State University
Omar El Ariss	The Pennsylvania State University, Harrisburg
Omar Elariss	
Mahmoud Elish	King Fahd University of Petroleum & Minerals
Davide Falessi	
Behrouz Far	University of Calgary
Liana Fong	IBM T. J. Watson Research
Fulvio Frati	Università degli Studi di Milano
Jerry Gao	San Jose State University
Kenan Gao	
Felix Garcia	Alarcos Research Group, Information and Systems Department, Escuela Superior de informática, University of Castilla-La Mancha
	University of Castilla-La Mancha
Ignacio García	Universidad Politecnica de Madrid
Raúl García-Castro	
Swapna Gokhale	
Wolfgang Golubski	Westfälische Hochschule Zwickau
Desmond Greer	
Christiane Gresse von Wangenheim	Federal University of Santa Catarina - UFSC
Katarina Grolinger	UWO
Hassan Haghighi	Shahid Beheshti University
Hao Han	Kanagawa University
Xudong He	Florida International University
Miguel Ángel Herranz	UAH
Rubing Huang	
Shihong Huang	Florida Atlantic University
Bassey Isong	
Clinton Jeffery	University of Idaho
Yue Jiang	Fujian Normal University
Jason Jung	Yeungnam University
Selim Kalayci	East Tennessee State University
Marcos Kalinowski	Universidade Federal Fluminense
Eric Kasten	
Taghi Khoshgoftaar	Florida Atlantic University
Claudiu Kifor	
Jun Kong	North Dakota State University
Aneesh Krishna	Curtin University, Australia
Vinay Kulkarni	Tata Consultancy Services
Yu Lei	University of Texas at Arlington
Meira Levy	Shenkar Engineering, Design, Art
Bixin Li	SOUTHEAST UNIVERSITY
Ming Li	National Lab for Novel Software Technology, Nanjing University
Xin Li	
Yuan-Fang Li	Monash University
Zhi Li	College of Computer Science and Information Technology, Guangxi Normal University

Jianhua Lin	
Frank Liu	Missouri University of Science and Technology
Shih-Hsi Liu	California State University, Fresno
Ting Liu	Xi'an Jiaotong University
Xiaodong Liu	Edinburgh Napier University
Yi Liu	
Luanna Lopes Lobato	Universidade Federal de Pernambuco - UFPE
Baojun Ma	Beijing University of Posts and Telecommunications
Ivan Machado	UFBA - Federal University of Bahia
Marcelo Maia	Federal University of Uberlândia
Riccardo Martoglia	FIM - University of Modena
Beatriz Marín	Universidad Diego Portales
Santiago Matalonga	Universidad ORT Uruguay
Hong Mei	Peking University
Hsing Mei	Fu Jen Catholic University
Andre Menolli	Universidade Estadual do Norte do Paraná - UENP
Ali Mili	NJIT
Alok Mishra	Atilim University, Incek 06836, Ankara - Turkey
Manuel Mora	Autonomous University of Aguascalientes
Kia Ng	ICSRiM - University of Leeds
Allen Nikora	
Edson Oliveirajr	State University of Maringá
Xin Peng	Fudan University
Oscar Pereira	University of Aveiro
Dragutin Petkovic	San Francisco State University
Antonio Piccinno	University of Bari
Alfonso Pierantonio	University of L'Aquila
Daniel Plante	Stetson University
Rick Rabiser	Christian Doppler Laboratory for Monitoring and Evolution of Very-Large-Scale Software Systems, Johannes Kepler University
Filip Radulovic	Ontology Engineering Group, Universidad Politécnica de Madrid
Damith Rajapakse	National University of Singapore
Rajeev Raje	IUPUI
Henrique Rebêlo	Federal University of Pernambuco - UFPE
Marek Reformat	
Robert Reynolds	Wayne State University
Ivan Rodero	
Elder Rodrigues	Pontifical Catholic University of RS - PUCRS
Daniel Rodriguez	The University of Alcalá
Samira Sadaoui	University of Regina
Seyed Masoud Sadjadi	Florida International University
Farshad Samimi	Enphase Energy
Claudio Sant'Anna	Federal University of Bahia
Akila Sarirete	
Andreas Schoenberger	Distributed and Mobile Systems Group - University of Bamberg

Abdelhak Seriai	Lirimm/université Montpellier 2
Michael Shin	
Martin Solari	Universidad ORT Uruguay
Qinbao Song	Xi'an Jiaotong University
George Spanoudakis	Department of Computer Science, City University
Jing Sun	The University of Auckland
Yanchun Sun	School of Electronics Engineering and Computer Science, Peking University, China
Gerson Sunyé	Université de Nantes
Chuanqi Tao	NJUST
Jeff Tian	Southern Methodist University
Genny Tortora	Department of Management and Information Technology - University of Salerno
Genny Tortora	
Mark Trakhtenbrot	Holon Institute of Technology
Peter Tröger	Hasso Plattner Institute, University of Potsdam
T.H. Tse	The University of Hong Kong
Burak Turhan	University of Oulu
Christelle Urtado	LGI2P - Ecole des Mines d'Alès
Sylvain Vauttier	LG2IP / Ecole des Mines d'Alès
Silvia Vergilio	
Sergiy Vilkomir	East Carolina University
Aaron Visaggio	University of Sannio
Arndt Von Staa	
Gursimran Wallia	North Dakota State University
Huanjing Wang	Western Kentucky University
Jiacun Wang	
Linzhang Wang	Nanjing University
Xiaoyin Wang	University of Texas at San Antonio
Ye Wang	Zhejiang Gongshang University
Yong Wang	
Hironori Washizaki	Waseda University
Victor Winter	University of Nebraska at Omaha
Guido Wirtz	University of Bamberg
Eric Wong	University of Texas at Dallas
Franz Wotawa	Technische Universität Graz
Dianxiang Xu	Boise State University
Frank Xu	
Haiping Xu	University of Massachusetts Dartmouth
Zhiwei Xu	University of Michigan - Dearborn
Guowei Yang	Texas State University
Hongji Yang	Bath Spa University
Huiqun Yu	
Du Zhang	California State University
Yong Zhang	Web and Software Technology R&D center, Tsinghua University, Beijing, P.R.China
Yunpeng Zhang	
Zhenyu Zhang	

Jiang Zheng
Jianlin Zhu
Xingquan Zhu
Eugenio Zimeo

ABB Inc., US Corporate Research
South-Central University for Nationalities
Florida Atlantic University
University of Sannio

External Reviewers' Index

Aktouf, Oum-El-Kheir
Aranda López King, Alejandrina María
Ardito, Carmelo
Arimoto, Mauricio
Assunção, Wesley K. G.

Bai, Xiaoying
Barat, Souvik
Bhattacharya, Swapan
Bianchi, Alessandro

Cao, Buqing
Castro Llanos, John Wilmar
Courbis, Anne-Lise

Dittman, David
Du, Weichang
Duan, Feng

Fang, Chunrong
Fazelpour, Ali
Ferreira Aranda, Juan Marcelo
Freitas Duarte Filho, Nemesio
Fu, Yujian

Ghandehari, Laleh
Guizzo, Giovanni

Ha, Hsin-Yu
Hao, Dan
Harispe, Sébastien
Hayrapetian, Allenoush
He, Tiantian
Hendijani Fard, Fatemeh
Huang, Tian
Hwa, Jimin

Kane, Shridhar

Lanzilotti, Rosa
Li, Bing
Li, Ge
Liu, Yuzhen

Macchi, Darío
Malhotra, Ruchika
Marcolino, Anderson
Matturo, Gerardo
Mohamed, Emad Amin
Mokni, Abderrahman
Moraga, Ma Ángeles

Neves Esteca, Antonio Marcos
Nusayr, Amjad

Park, Jihun
Phadke, Aboli
Pouyanfar, Samira

Quintana, Gerardo

Rehman, Zobia
Rodríguez, Francy D.
Roychoudhury, Suman
Rybarczyk, Ryan

Schoenberger, Andreas
Seo, Dongwon
Shi, Qingkai
Shin, Donghwan
Sun, Chenglong
Sun, Xiaobing
Sunkle, Sagar

T. Geraldi, Ricardo
Teixeira, Leopoldo
Tian, Haiman
Tieke, He

Vessio, Gennaro

Wang, Jiacun
Wang, Yong
Wen, Wanzhi

Yang, Yimin

Zhang, Han
Zhang, Long
Zhang, Tao
Zhang, Wei
Zhang, Ye

Zhang, Zhiqiang
Zhao, Haiyan
Zhao, Junfeng
Zhou, Peiyuan