An Initial Examination of Computer Programs as Creative Works

Trina C. Kershaw[1], Ralph D. Clifford[2], Firas Khatib[3], and Adnan El-Nasan[3]

Department of [1]Psychology, [2]School of Law, and [3]Department of Computer and Information Science, University of Massachusetts Dartmouth

Author Note

Firas Khatib https://orcid.org/0000-0002-6817-7297

Correspondence concerning this article should be addressed to Trina C. Kershaw, Department of Psychology, University of Massachusetts Dartmouth, 285 Old Westport Road, North Dartmouth, MA 02747. Email: tkershaw@umassd.edu

Abstract

Products from many domains (art, music, engineering design, literature, etc.) are considered to be creative works, but there is a misconception that computer programs are limited by set expressions and thus have no room for creativity. To determine whether computer programs are creative works, we collected programs from 23 advanced graduate students that were written to solve simple and complex bioinformatics problems. These programs were assessed for their variability of expression using a new measurement that we designed. They were also evaluated on several elements of their creativity using a version of Cropley and Kaufman's (2012) Creative Solution Diagnosis Scale that was modified to refer to programming. We found a high degree of variation in the programs that were produced, with 11 unique solutions for the simple problem and 20 unique solutions for the complex problem. We also found higher ratings of propulsion-genesis and problematization for the complex problem than for the simple problem. This combination of variation in expression and differences in level of creativity based on program complexity suggests that computer programs, like many other products, count as creative works. Implications for the creativity literature, computer science education, and intellectual property law, particularly copyright, are discussed.

Keywords: creativity, computer programming, copyright law, creative products

An Initial Examination of Computer Programs as Creative Works

> A common lay belief is that programming takes place in highly structured environments, relying solely on formal languages and standard techniques, with little or no room for creativity. (Kozbelt et al., 2012, p. 58)

Common definitions of creative works are that they are novel and useful (Sternberg & Lubart, 1999) or novel, effective, and whole (Wieth & Francis, 2018), what are known as standard definitions of creativity (Runco & Jaeger, 2012). In technical fields including engineering and computer programming, creators may emphasize the functionality of their work over its originality or aesthetic value (cf. Cropley & Kaufman, 2019). As Kozbelt et al.'s (2012) quote illustrates, there is a concern that products from some domains, such as computer programming, are not considered creative works. This exclusion is not without consequence, particularly in intellectual property law (IP).

For computer programs, the law predominantly uses two IP systems to protect ownership -- patents and copyrights – but only the latter directly involves evaluations of creativity. IP makes the creativity of a program directly relevant to its copyrightability (Clifford, 2004; *Feist Publications, Inc. v. Rural Telephone Service Company, Inc.,* 1991) but does not care about that for its patentability (*Graham v. John Deere Co.*, 1966). At best, creativity is only indirectly referenced in patent law because the basic requirements for a patent are that the invention must be a "new and useful" and not an "obvious" extension of existing inventions (Patent Act of 1952, §§ 101-103). The court need not determine that the invention itself is "creative" or even that it resulted from a creative process by its human inventor; instead, it compares the newly claimed invention against all prior ones (called the "prior art") to make sure that it is new and nonobvious (see *Eibel Process Co. v. Minnesota & Ontario Paper Co.*, 1923; *Graham v. John Deere Co.*, 1966). Although one can easily posit that "creativity" is likely to be the driving force that allows

the creator to invent what did not previously exist, the law does not care about the motivation or source of the invention so long as it is appropriately different from the prior art. The U.S. Supreme Court expressly recognized this in the *Graham* case when it acknowledged that Congress, by adopting § 103, was repudiating the "flash of creative genius" test that had been required before the 1952 Patent Act was adopted (*Graham v. John Deere Co.*, 1966, at 15).

Copyright law differs. The courts are required to find that a "modicum of creativity" led to the expression contained in the work as the basic standard for determining copyright eligibility (*Feist Publications, Inc. v. Rural Telephone Service Company, Inc.,* 1991, at 346 & 362; see also Clifford, 1997). In doing so, however, they have shown an inconsistent understanding of the nature of creativity underlying expressive works (Clifford, 2004; compare *Boisson v. Banion, Ltd.,* 2001 with *Satava v. Lowry,* 2003). When computer programs are the subject of the modicum standard or its derivatives such as the idea-expression merger or filtration concepts (e.g., *Computer Assoc. Int'l, Inc. v. Altai, Inc.*, 1992), the courts' decision-making becomes even more indeterminate (*Google LLC v. Oracle America, Inc.*, 2021, at 1197 (declining to address whether Oracle's APIs are subject to copyright because, if they are, Google's use of them was fair under the Copyright Act of 1976, § 107)). Of course, with little scientific evidence of whether programs are creative products, applying the modicum of creativity standard is unlikely to be based on fact.

Our goal in the following paper, therefore, is to establish that computer programs are creative works. Beyond the direct scientific consequences of expanding knowledge about human creativity, the practical effect of establishing this will be significant as the distractive and expensive litigation over program creativity can be minimized.

What Makes a Work Creative?

Following Rhodes' (1961) description of the 4 Ps of creativity, our focus is on the

creative product. In psychology, creative products can take the form of responses on standard

divergent tests such as the Torrance Tests of Creative Thinking (Torrance, 1974) or the

Alternative Uses Task (Guilford, 1967), or can be domain-specific products, such as artistic

works, musical scores, or engineering designs. Our focus is on domain-specific products because

evaluation of these products is the "gold standard" of creativity assessment (Plucker & Makel,

2010).

What makes a product creative differs on the assessment that is used, although many

researchers follow the standard definition (Runco & Jaeger, 2012) that creative products are

novel and appropriate. The most popular assessment is Amabile's (1982) consensual assessment

technique (CAT), which involves a global rating of creativity by a group of experts within a

particular domain. Amabile (1982) argued that the raters' experience within the domain would

guide them to similar decisions regarding the creativity of a product (see Baer & McKool, 2009,

for a review of the CAT). Unfortunately, the CAT is often used to evaluate products generated

by individuals with low domain knowledge, such as children or undergraduates from a research

pool (cf. Amabile, 1982; Kaufman et al., 2013, Study 1; Storme et al., 2014), although there are

exceptions in studies that focus on advanced students or domain experts (cf. Beaty et al,, 2013;

Dunbar, 1997; Getzels & Csikzentmihalyi, 1976).

Other measurements have been developed to assess creative products generated by

individuals with high domain knowledge, such as advanced engineering students or design

professionals. For example, Charyton et al. (2008) developed the Creative Engineering Design

Measurement (CEDA) to evaluate product design by using a team of expert raters. Similarly to

the CAT, the CEDA asks judges to make an overall rating of a product's creativity. An overall

rating of creativity was also used by Kershaw et al.'s (2019) Decision Tree for Originality

Assessment in Design (DTOAD) to assess concepts that are produced during the ideation stage

of engineering design, but Kershaw et al. (2019) used trained coders instead of domain experts.

In contrast, Cropley and Kaufman's (2012) Creative Solution Diagnosis Scale (CSDS) consists

of a series of statements allowing for the evaluation of a creative product's relevance and

effectiveness, problematization, propulsion, elegance, and genesis. Not only does the CSDS

capture novelty and appropriateness of the product, it also captures the aesthetic components of

the product, which are important for evaluation of creativity in multiple domains, including

computer programming (Kozbelt et al., 2012). In addition, the CSDS is a reliable, valid metric

for the assessment of functional creativity, which focuses on novel products that serve a useful

purpose. Several previous studies have indicated high inter-rater agreement with the CSDS

(Cropley & Cropley, 2016; Cropley et al., 2011), and have indicated that self-ratings and expert

ratings are similar (Kaufman et al., 2016). Our review of these different methodologies led us to

adopt the CSDS as the primary evaluative technique for this study. Computer programs are

clearly functional products which matches the test's intended target.

Beyond novelty and appropriateness, variation in expression is also an important

characteristic of creative products. The idea of variability contributing to creativity has a long

history within creativity research, beginning with Campbell's (1960) assertion that variability in

behavior is necessary to discover novel solutions to problems (see also Simonton, 1999). Stokes

(2001) has argued that high variability levels are a hallmark of divergent thinking and that

variability within a domain can be learned during the early stages of skill acquisition (Stokes,

1999). In addition to variability affecting the production of creative products, greater variability

is also associated with higher judgments of creativity (Young & Racey, 2009).

Variability is not necessarily just blind variation (cf. Campbell, 1960; Simonton, 1999), however. Rather, constraints are necessary to generate and sustain novelty (Stokes, 2007). Stokes' (2007) constraint model of novelty proposes paired constraints which simultaneously limit search within a problem space to known solutions and expand search to unfamiliar solutions. Stokes (2007) provides support for this model through an analysis of Pop Art, but support for this model can also be found within functional creativity domains, such as engineering and computer programming. Cropley et al.'s (2017) description of functional creativity suggests that engineers must work within the constraints imposed by the intended function of their designs. Working from previous ideas (constrained to known solutions) is a common approach in functional creativity; Sternberg et al. (2002) suggest six ways in which creativity can propel a field by working from existing knowledge. Likewise, in computer programming, a programmer must meet constraints based on the programming language or computer system being used or on requirements from the problem or client, yet also can promote innovation within the field by choosing novel approaches for solving the particular processing needs or by writing implementations of known approaches in unconstrained ways.

Creativity in Computer Programming

Programmers believe that they engage in creative acts by writing computer programs (cf. Glass, 2006; Knobelsdorf & Romeike, 2008). One of the most instrumental founders of academic computer science, Donald E. Knuth, argued that programming is "…an aesthetic experience much like composing poetry or music" (Knuth, 1973, p. v) and that computer programming was an art, rather than a science (Knuth, 1974). While aesthetics is important for creativity in many domains (cf. Kozbelt et al., 2012), there is unfortunately a dearth of empirical research on the creative process of programming or programs as creative products.

There is scant existing literature on creativity within computer programming, or more broadly within computer science. One review chapter primarily focused on a historical review of breakthrough innovations within the field (Saunders & Thagard, 2005). While the examination of case studies of Big C creativity is a common approach within the creativity literature (cf. Simonton, 2010), this methodology does not compare related products and cannot always reveal the process that led to a creative product, especially in the absence of documentation of the process by the creator (a concern raised by Barnett & Romeike, 2017). Another review chapter went beyond historical cases to discuss fundamental issues such as defining how creativity manifests within components of computer science, but again, does not provide any empirical data about the process or programs as creative products (Barnett & Romeike, 2017).

Two papers by Kozbelt and colleagues provided some empirical insight into the role of creativity in programming. Kozbelt et al. (2012) found that programmers did have aesthetic experiences while writing code, and that they believed aesthetics was an important characteristic of code (although not as important as the program's functionality). Kozbelt et al. (2015) examined the creative process using verbal protocols that were collected while programmers improved a poorly-written program. Examination of these protocols revealed a higher level of secondary process thought (abstraction, social behavior, instrumental behavior, restraint, etc.) than the protocols of artists completing a drawing task, and a lower amount of emotion-related words than the artists. These papers were an important step in beginning to examine creative beliefs and the creative process within computer programming. However, they still do not establish that programs are creative works.

The Current Study

Our current study had the goal of establishing that computer programs are creative works. Given the dearth of research on creativity within computer programming, particularly on evaluation of programs, this study provides an important first step in the literature. Drawing from previous research establishing the novelty, appropriateness, aesthetic value, and variability that are common to creative products, we chose two ways of assessing computer programs. First, we chose the CSDS (Cropley & Kaufman, 2012), a reliable, valid instrument for measuring the relevance and effectiveness, problematization, propulsion, elegance, and genesis of programs.

Second, as an adjunct to the CSDS, we developed a way to measure variability within computer programs (the Program Control and Descriptive Variables assessment, or PCDV). Most computer programs, including all of those within the study's dataset, are written using a procedure-oriented language (see Sammet & Hemmendiger, 2000). There are many of these languages that have been created, but modern versions contain universally used control constructs to determine how the program functions (Cox & Hemmendiger, 2000). Because of the commonality in statement types, a program can be broken down into a single set of programming control and descriptive variables based on the number of times each these universally used coding statement was used regardless of the name that the particular language has chosen for the statement (some languages call an automatic loop with a counter a "*for*" loop, while others call it a "*do*" or "*perform*" loop, for example). Then, by concatenating the count of each type of statement together, a single descriptor code could be expressed that captures the algorithmic and expressive essence of a program. The PCDV allowed us to measure how much significant variation was found among the programming examples we studied.

These assessments were applied to computer programs generated by computer science graduate students. All these students were competent programmers; indeed, they are

appropriately considered quasi-experts in the field. These coders each produced two programs which were designed to solve a simpler and a more complex bioinformatics problem. Other than needing to produce functioning programs that operated sufficiently quickly to process the large bioinformatic dataset within five minutes, the programmers were not provided with any additional constraints such as the length of the program or the programming language to use.

Because variability is a hallmark of creative behavior (Campbell, 1960; Simonton, 1999; Stokes, 1999, 2001), we expected to see a wide variety of creative expressions within the programming choices made by the research subjects. In addition, we hypothesized greater variability for the more complex problem. Further, we predicted that there would be greater levels of novelty and aesthetic quality for the complex problem than for the simple problem. We did not expect any differences between the problems in their level of appropriateness (relevance and effectiveness, according to the CSDS) given the constraint that students had to produce functional programs.

Method

Participants

The programs used in this study were created by 29 graduate students who were enrolled in an Advanced Bioinformatics course at the University of Massachusetts Dartmouth during the Spring 2019 semester. One student was pursuing a Chemistry and Biochemistry Ph.D., another a Data Science Ph.D., and the remaining twenty-seven students were part of a Computer and Information Science M.S. program. The majority of the M.S. students were international students, but no additional demographic data were collected about the students. The prerequisite for this course included demonstrated competence in algorithms and data structures, so all students already knew how to code. Consequently, the lectures in the course focused on

bioinformatics while the assignments required the coding of relevant algorithms and no in-class

time was necessary to teach programming. Because our data were collected from course

assignments, we received exempt approval from the University's I.R.B.

Some of the data provided by the students were excluded from the study. First, two

students did not submit their code via the course management software. Second, four students

submitted a program that was not functional. In both cases, these participants' data were

excluded from the study, leaving us with 23 proficient programmers.

Materials

Problems

Students were provided with different bioinformatics homework problems throughout the

semester from the *Rosalind* website (http://rosalind.info/problems/locations/), including a

simpler problem (http://rosalind.info/problems/ba1g) as well as a more complex one

(http://rosalind.info/problems/ba3h). *Rosalind* was chosen because it accompanies the textbook

for this Advanced Bioinformatics course and has specific problems that are discussed in each

chapter: http://rosalind.info/problems/list-view/?location=bioinformatics-textbook-track. Another

advantage of this website is that it will confirm if a solution is correct, and submissions can be

made using any programming language. Programs also need to be reasonably quick as students

are only given five minutes to run their algorithm and submit their answer, given a large

bioinformatics dataset as input.

The simple problem, *Compute the Hamming Distance Between Two Strings*, takes two

different strings of DNA as input (such as GGGCCGTTGGT and GGACCGTTGAC) and

outputs the number of mismatches between them. This algorithm is particularly useful when

searching for mutations in DNA strings. In the above example, the correct answer is three, as

they have three differences between them (**bolded** below):

GG**G**CCGTTG**GT**
GG**A**CCGTTG**AC**

The complex problem, *Reconstruct a String from its k-mer Composition*, takes in an

integer *k* as input along with a set of *k*-mers of *Patterns*. For example: CTTA, ACCA, TACC,

GGCT, GCTT, and TTAC would have *k* equal to four since these are 4-mers (they have length

four). The goal of this more complicated problem is to reconstruct a string from its *k*-mer

composition, where the *k*-mers overlap with one another by *k-1* letters (three letters in our

example: **GGCT** and **GCT**T both have **GCT** in common). This algorithm is important for

genome assembly where long strands of DNA have been fragmented into shorter pieces (*k*-mers).

Looking at the example above, the solution would be the following string: GGCTTACCA.

Specifically, six different 4-mers can reconstruct that string as follows:

```
GGCT
 GCTT
  CTTA
   TTAC
    TACC
     ACCA
```

Creative Solution Diagnosis Scale (CSDS)

The CSDS (Cropley & Kaufman, 2012) is a scale designed to provide consensual

assessment for functional creativity using non-expert judges. The scale shows a high degree of

reliability (Cronbach's alpha = .96). It is composed of five subscales corresponding to five

factors identified by Cropley and Kaufman (2012): *relevance and effectiveness*,

problematization, *propulsion*, *elegance*, and *genesis*. We used the 21-item version of the CSDS

(Cropley & Cropley, 2016) and modified the items for the assessment of computer programs to

use the phrase *the program* rather than *the output*. For example, item 1 from the relevance and effectiveness subscale was reworded from "The output accurately reflects conventional knowledge and/or techniques" to "The **program** accurately reflects conventional knowledge and/or techniques."

Procedure

Students from the graduate Advanced Bioinformatics course completed the programs as homework assignments for their course, where they were given two weeks for each assignment. The simple problem was given earlier in the semester, whereas the complex one was given later in the course. Once students successfully submitted their correct program, they were required to upload and share their code with other students on the *Rosalind* website. This code was only accessible to students who successfully submitted that specific *Rosalind* problem.

Students provided self and peer ratings of the programs, using the CSDS as part of a class assignment. So as to not add the pressure of rating on top of having to complete a functional program, students provided the ratings as part of the assignment subsequent to the one in which they completed one of the target programs. They were instructed to complete a multiple-choice survey about the code and were provided with the statements from our program-focused version of the CSDS as part of their instructions. For peer ratings, students were asked to rate programs produced by the student before and after them on a class list. Thus, programs were rated by different peers; each student only rated two programs. Having multiple raters only evaluate a selection of products is a procedure followed in other research (cf. Green et al., 2014; Kudrowitz & Wallace, 2013; Runco et al., 1994) to reduce the potential of rater fatigue in comparing creative products (cf. Cseh & Jeffries, 2019).

The expert rater, the fourth author, is a computer science faculty member with a computer engineering-based Ph.D. having approximately 17 years of experience in the computer industry. He did not teach the course from which the data were obtained. First, the expert rater reviewed the CSDS instrument with the first and third authors and reviewed the assignment instructions given to the students with the third author. He then completed CSDS ratings for every program.

Analysis

Program Control and Descriptive Variables (PCDV)

The second author developed a coding scheme to describe how a programmer chose to implement a problem solution that could provide an objective estimate of fluency within the programs. These programming control and descriptive variables were chosen because they would capture the overall structure of the software based on determining the number of times each universal-used programming control methodology had been used by the programmer, while excluding trivial variations such as the program's variable names. Further, the PCDV functions regardless of the source language used by the programmer so long as it is a modern procedure-based language. Table 1 shows each variable and how it was operationally defined. All defined variables were used except the *case* variable which was dropped from further analysis because it was not present in the data.

Two research assistants viewed the coding scheme with the authors, then applied it to the programs generated for the simple and complex problems ($n = 23$). Inter-rater agreement was calculated using intra-class correlations; a mean-rating ($k = 2$), absolute agreement, two-way mixed effects model was applied. Average ICC for the simple problem was .94 (95% CI = .88-.97) and average ICC for the complex problem was .99 (95% CI = .986-.998). Any coding disagreements were resolved by the third author.

Once each of the programs had been analyzed by the research assistants, a single descriptive code was created. The program's code is a sequence of digits created by concatenating the number of times each programming control and descriptive variable was used in the program. For our study, as no programming control statement was used more than 99 times within a program, each of the seven programming control statements described in Table 1 is represented by the necessary two digits giving it a possible count range from *not used* (e.g. 00) to *used* for up to 99 times. The two digits that represent each programming control variable have a fixed relative location in the overall 14-digit descriptive code that represents the program and captures its structure. Table 2 shows the order of the programming controls, examples of their repetitions for different program samples, and the resulting PCDV for each sample. The Appendix contains a demonstration of the PCDV coding scheme for four sample programs that solve the simple problem used in our study.

The final step in the analysis was to determine the number of unique descriptive codes found within the simple and complex data sets. Because the descriptive code for two programs would be the same if the code was substantially identical, a difference in the code indicates the presence of some form of expressive variation within the program samples, a relevant measure under current legal precedents such as *Feist Publications, Inc. v. Rural Telephone Service Company, Inc.* (1991; see also Clifford, 1997; 2004; 2018). To measure the approximate size of these variations, the number of unique descriptive codes was divided by the sample size to create a variation statistic which could range from $1/n$ (which would indicate no variation) and 1.000 (which would indicate complete variation). The range of values of the variation statistic indicates increasing differences in coding as it goes from small to large.

CSDS Scoring

As described in the Procedure, CSDS scores were provided for each program by the student who wrote the program (self rating), at least one classmate (peer rating), and the fourth author (expert rating). Different types of raters have been used in past research (Moneta et al., 2010). The inter-rater consistency, scale validity, and factor structure of the CSDS were assessed following the procedures used by the original authors of the instrument, Cropley and Kaufman (2012).

Inter-Rater Reliability. Inter-rater reliability was assessed using Cronbach's (1951) coefficient alpha, following the guidelines established by Cropley and Kaufman (2012). For the simple problem, we had 74 valid ratings (17 self, 34 peer, and 23 expert ratings). Ten additional ratings (3 self and 7 peer) were removed due to low item-total correlations or lack of variation in the individual's ratings (ex. choosing 0s for every item). For the complex problem, we had 72 valid ratings (17 self, 35 peer, and 20 expert ratings). Sixteen additional ratings (3 self, 10 peer, and 3 expert) were removed due to low item-total correlations or lack of variation in the individual's ratings. The average Cronbach's coefficient alpha was .70 (range .36-.92), which is an acceptable level (Nunnally & Bernstein, 1994; Tavakol & Dennick, 2011). For comparison, past research has reported inter-rater reliabilities between expert and novice raters ranging from $\kappa = .2-.9$ (Green et al., 2014) and between self, peer, and expert raters ranging from $r = .27-.56$ (Moneta et al., 2010). Based on the acceptable level of inter-rater reliability that we obtained, and on procedures used in previous research (Green et al., 2014; Moneta et al., 2010; Runco et al., 1994), self, peer, and expert ratings were averaged to create one score for each item for each program.

Scale Reliability. Scale reliability was assessed using several measures. Following the guidelines from Cropley and Kaufman (2012), we report Cronbach's (1951) coefficient alpha.

Given the issues with using coefficient alpha as a measure of scale reliability (cf. Bendermacher, 2010; Dunn et al., 2013; Graham, 2006), and the correlations among the scale factors described below, we also report Guttman's (1945) λ_2 and McDonald's (1999) $\acute{\omega}$. Scale reliability was calculated for each problem based on the valid CSDS ratings across the twenty-one CSDS items. Based on the 74 valid ratings for the simple problem, we obtained the following reliability estimates: $\alpha = .94$, $\lambda_2 = .95$, $\acute{\omega} = .95$. Based on the 72 valid ratings for the complex problem, we obtained the following reliability estimates: $\alpha = .95$, $\lambda_2 = .95$, $\acute{\omega} = .95$. These obtained values were nearly identical and indicate excellent scale reliability (Nunnally & Bernstein, 1994; Tavakol & Dennick, 2011; Watkins, 2017).

Factor Structure. Following Cropley and Kaufman (2012), a principal axis factor analysis was conducted on the 21 items of the revised CSDS (Cropley & Cropley, 2016) with oblique rotation (direct oblimin) to allow the factors to correlate. The number of factors was set to five, as Cropley and Kaufman (2012) found five factors. For the purposes of establishing the factor structure of the CSDS within our dataset, and to ensure sufficient data for analysis, ratings for both the simple and complex problem were included. The Kaiser-Meyer-Olkin Measure of Sampling Adequacy for the analysis was .93, and Bartlett's Test of Sphericity resulted in χ^2 (210, N = 146) = 2819.87, $p < .001$, indicating that the data were suitable for factor analysis.

Five identifiable components were extracted that corresponded to the five factors identified by Cropley and Kaufman (2012), accounting for 74.95% of the variance in the CSDS data. Table 3 shows the factor loadings after rotation. The items that cluster on the same factor indicate that factor 1 represents how well the program applies to other situations (*genesis*), factor 2 represents the beauty and cohesiveness of the program (*elegance*), factor 3 represents how well the program displays knowledge and satisfies the requirements (*relevance and effectiveness*),

factor 4 represents how well the program draws attention to problems in what already exists (*problematization*), and factor 5 represents how well the program propels the field (*propulsion*). As shown in Table 3, factor 5 (*propulsion*) had an eigenvalue of .41 and a high correlation of $r =$ -.78 with factor 1 (*genesis*). Due to the multicollinearity between these variables and the convergence of the scree plot for a four-factor solution, factors 1 and 5 were averaged into a single factor (*propulsion-genesis*) for further analyses.

<div align="center">

Results

</div>

Program Control and Descriptive Variables (PCDV)

A series of paired-samples t-tests were conducted to examine differences between the simple and complex problems. As shown in Table 4, the complex problem contained more *subroutines*, *for* loops, *while* loops, *if* statements, *else* statements, and *goto* statements. These increasing values from a simple to a more complex problem are consistent with a non-empirical evaluation that would posit that more program control constructs would be needed in a more complex program.

As described in the Method section, a PCDV variation statistic was calculated based on the descriptive code assigned to each program within the simple and complex data sets. For the simple programs, with 11 unique solutions found within the 23 programs submitted, the statistic was 0.478 out of a range between 0.043 (all programs have the same descriptive code) and 1.000 (no two programs have the same descriptive code). This figure is surprisingly high considering the relative simplicity of the code being produced. Not unexpectedly, as the second program was computationally more complex than the first, the variation statistic for it was much higher at 0.870 (out of a range between 0.043 and 1.000), with 20 unique solutions found within the 23 programs submitted.

Creative Solution Diagnosis Scale (CSDS)

We conducted a series of paired-samples t-tests comparing the simple and complex problems on the four facets of the CSDS: propulsion-genesis, elegance, relevance and effectiveness, and problematization. As shown in Table 5, the problems were significantly different on two of these factors, with the complex problem having higher ratings than the simple problem for propulsion-genesis and problematization, and marginally higher ratings for elegance (due to the Bonferroni correction for multiple comparisons; Field, 2017). The problems did not significantly differ for ratings of relevance and effectiveness.

Discussion

We found greater variability in problem solutions for the complex problem than the simple problem, in support of our first hypothesis. Of our 23 participants, 11 unique programs were produced for the simple problem and 20 unique programs were produced for the complex problem. This result demonstrates that there is a large variation of programming expressions that can be used to solve even simple coding problems. For more complex programs, almost every version created was measurably different from the others. Because the programs within each data set solved an identical problem and had been shown to function correctly, the differences in the coding solutions cannot be due to a need that is dictated by the algorithm being implemented. We believe that the variations found are due to the exercise of individual creativity by the different programmers.

We also found differences between the simple and complex problem on subjective ratings of creativity via the CSDS. As hypothesized, the complex problem had higher ratings of elegance and propulsion-genesis. The latter result represents the higher level of novelty we expected for programs that were produced to solve the complex problem and helps to establish that variation

in the generated solutions can be explained by programmer creativity. Also as hypothesized, we found no differences between the simple and complex problems on ratings of relevance and effectiveness. This is not a surprising result because participants were required to produce functional programs. Additionally, in computer science, as well as in engineering fields, functionality is considered to be a critical aspect of design (Cropley & Kaufman, 2019; Kozbelt et al., 2012).

Overall, our results provide initial support for the claim that computer programs are creative works. Programs are a form of functional creativity (Cropley et al., 2017) which are novel and appropriate. Like other creative works, programs have aesthetic value; indeed, as noted by Knuth (1974), computer programming is a form of art. Our results support the idea that even within structured environments, there is still room for creativity – the high degree of variation of expression seen within the programs in our study supports assertions that variability in behavior is a key contributor to creativity (Campbell, 1960; Stokes, 2001). Further, our results go beyond existing research on creativity in computer programming that has focused on historical cases (Saunders & Thagard, 2005) or the aesthetic experience of programming (Kozbelt et al., 2012).

While our research has helped to advance the currently scant literature on creativity within computer programming, there are several limitations to our work. First, the programs in our study were written by graduate students, who, although being fully competent programmers, may be at best considered quasi-experts in the field. For example, most are likely not to be fluent in multiple programming languages and may have yet to work within the large teams of programmers that are typical in the professional programming world. As shown in multiple studies in non-programming disciplines, people with expertise in a field approach problems

differently than less experienced practitioners (cf. Chi et al., 1981) and apply different perceptual

processes (cf. Chase & Simon, 1973). Level of expertise may affect creativity as well, including

how skilled performance is impacted by varying creativity instructions (Rosen et al., 2017).

Consequently, we cannot eliminate the possibility that our results were influenced by our

research subjects' level of experience rather than reflecting programming in general. However, it

is important to note that knowledge within computer science can become obsolete quickly due to

the demands of a dynamic discipline, and "expertise" may not be reflected in years of experience

(Sonnentag et al., 2006). Instead, the amount of experience, such as the number of programs

written, could be considered (cf. Rosen et al., 2017), or, as recommended by Sonnentag et al.

(2006), expertise in computer programming could be conceptualized as high performance rather

than years of experience. Future research should prescreen potential programmers for their level

of programming skill.

A second limitation is our use of a combination of self, peer, and expert ratings for the

CSDS. Although our level of inter-rater agreement was acceptable, and this procedure has been

followed in past research (Moneta et al., 2010), other research has shown mixed evidence as to

whether novices and experts rate creative products similarly. Some research has shown high

levels of agreement between experts and novices (Freeman et al., 2015), but sometimes

agreement is better when the novices have some prior knowledge (Kaufman et al., 2013, Study 1;

Plucker et al., 2009). Other studies have shown disagreement between experts and novices (Katz-

Buonincontro et al., 2020; Kaufman et al., 2013, Study 2). Most research on creativity uses

expert raters, and we recommend that practice is followed in future research, although experts

can disagree with each other as well (cf. Cseh & Jeffries, 2019; Jeffries, 2017).

A third limitation of the present research is that all programs were written in response to bioinformatics problems. Though unlikely, as bioinformatics problems are not the most representative tasks for computer programmers, coding them may involve different levels of creativity than other programming tasks. For future research, we recommend using problems that are more representative of typically produced software, such as optimization-based problems (often used for such things as controlling just-in-time inventory and package delivery) and data searching problems (commonly used in online searching, operating system control, and scheduling tasks).

A final limitation of our research is a focus on the creative product; in this case, the programs that were produced. As noted by Corazza (2016), a complete understanding of creativity needs to take into account the process that leads to a product. The drafts of code that our participants wrote, and the programming choices that they made, also have the potential to be creative. Beyond the measurement of programming creativity contained within the product that is captured by the PCDV and CSDS, we recommend that future research examines the creative process of computer programming. Future investigators can do so by using techniques such as building on Kozbelt et al.'s (2015) use of verbal protocols to record programmers' thoughts and choices online or through the use of questionnaires to capture programming decisions and the application of relevant mental processes.

Even when accounting for the limitations in the current study, important lessons can be derived from it. Our preliminary research suggests that common assumptions about how programmers work are inaccurate. If programmers are consistently demonstrating creativity, as our study suggests, the approaches used to teach students how to code should be examined to allow computer science education to enhance this creativity (cf. Good et al., 2016; Romeike,

2007, Xu et al., 2018). Similarly, as the management of creative-exercising workers differs greatly from the techniques used with employees in less inventive occupations (Brooks, 1995; Sawyer, 2001), business studies may be needed to establish the most appropriate management techniques for programmers.

Most importantly, as discussed above, the legal system is dependent on a finding of creativity in establishing copyright protection for computer programs (*Feist Publications, Inc. v. Rural Telephone Service Company, Inc.,* 1991; Clifford, 1997). The current study establishes that the *Feist* requirement of having multiple ways of expressing the material is satisfied by all software, excepting only non-realistically trivial programs such as the code that can print "Hello, world." Further, the discovery of the degree of variation in expression that is found in even the most basic computer software should foreclose the current inappropriate use of legal doctrines— such as the merger or singularity of efficiency doctrines that have been developed for computer programs in copyright law (*Computer Assocs. Int'l, Inc. v. Altai, Inc.,* 1992*)*—that are used to argue for an overly restrictive scope of protection for programs (*Google LLC v. Oracle America, Inc.*, 2021, at 1197*)*.

References

Amabile, T.M. (1982). Social psychology of creativity: A consensual assessment technique. *Journal of Personality and Social Psychology, 43*(5), 997-1013. https://doi.org/10.1037/0022-3514.43.5.997

Baer, J., & McKool, S.S. (2009). Assessing creativity using the consensual assessment technique. In C. Shreiner (Eds.), *Handbook of research on assessment technologies, methods, and applications in higher education* (pp. 65-77). Information Science Reference.

Barnett, P.J., & Romeike, R. (2017). Creativity within computer science. In J.C. Kaufman, V.P. Glăveanu, & J. Baer (Eds.), *The Cambridge handbook of creativity across domains* (pp. 299-322). Cambridge University Press.

Beaty, R.E., Smeekens, B.A., Silvia, P.J., Hodges, D.A., & Kane, M.J. (2013). A first look at the role of domain-general cognitive and creative abilities in jazz improvisation. *Psychomusicology, 23*(4), 262-268. https://doi.org/10.1037/a0034968 262-268

Bendermacher, N. (2010). Beyond alpha: Lower bounds for the reliability of tests. *Journal of Modern Applied Statistical Methods, 9*(1), 95-102. https://doi.org/10.22237/jmasm/1272687000

Boisson v. Banian, Ltd., 273 F.3d 262 (2d Cir. 2001), https://caselaw.findlaw.com/us-2nd-circuit/1306383.html

Brooks, F.P. (1995). *Mythical man-month.* Addison-Wesley Publishing Co.

Campbell, D.T. (1960). Blind variation and selective retention in creative thought as in other knowledge processes. *Psychological Review, 67*(6), 380-400. https://doi.org/10.1037/h0040373

Charyton, C., Jagacinski, R. J., & Merrill, J. A. (2008). CEDA: A research instrument for

    creative engineering design assessment. *Psychology of Aesthetics, Creativity, and the*

    Arts, 2(3), 147–154. https://doi.org/10.1037/1931-3896.2.3.147

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4*(1), 55–81.

    https://doi.org/10.1016/0010-0285(73)90004-2

Chi, M.T.H., Feltovich, P.J., Glaser, R. (1981). Categorization and representation of physics

    problems by experts and novices. *Cognitive Science, 5*(2), 121-152.

    https://doi.org/10.1207/s15516709cog0502_2

Clifford, R.D. (1997). Intellectual property in the era of the creative computer program: Will the

    true creator please stand up? *Tulane Law Review, 71*, 1675-1703.

    https://scholarship.law.umassd.edu/fac_pubs/77/

Clifford, R.D. (2004). Random numbers, chaos theory and cogitation: A search for the minimal

    creativity standard in copyright law. *Denver Law Review, 82*(2)*,* 259-299.

    https://scholarship.law.umassd.edu/fac_pubs/84/

Clifford, R.D. (2018). Creativity revisited. *IDEA* 59(1), 25-39.

    https://scholarship.law.umassd.edu/fac_pubs/196/

Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 707-10 (2d Cir. 1992),

    https://casetext.com/case/computer-associates-intern-inc-v-altai-inc-9

Copyright Act of 1976, 17 U.S.C. § 101 *et seq.* (1976).

    https://uscode.house.gov/view.xhtml?path=/prelim@title17/chapter1&edition=prelim

Corazza, G.E. (2016). Potential originality and effectiveness: The dynamic definition of

    creativity. *Creativity Research Journal, 28*(3), 258-267. https://doi.org/

    10.1080/10400419.2016.1195627

Cox, T.L. & Hemmendiger, D. (2000). Procedure-oriented languages. In A. Ralston, E.D. Reilly, & D. Hemmendinger (Eds.), *Encyclopedia of computer science,* 1470-1475 (4th ed., pp. 1441-1443). Wiley.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrica, 16*(3), 297–334. https://doi.org/10.1007/BF02310555

Cropley, D.H., & Cropley, A.J. (2016). Promoting creativity through assessment: A formative computer-assisted assessment tool for teachers. *Educational Technology Magazine, 56*(6), 17-24. https://www.jstor.org/stable/44430503

Cropley, D.H., Cropley, A.J., & Sandwith, B.L. (2017). Creativity in the engineering domain. In J.C. Kaufman, V.P. Glaveanu, & J. Baer (Eds.), *The Cambridge handbook of creativity across domains* (pp. 261-275). Cambridge University Press.

Cropley, D.H., & Kaufman, J.C. (2012). Measuring functional creativity: Non-expert raters and the Creative Solution Diagnosis Scale. *The Journal of Creative Behavior, 46*(2), 119-137. https://doi.org/ 10.1002/jocb.9

Cropley, D.H., & Kaufman, J.C. (2019). The siren song of aesthetics? Domain differences in creativity in engineering and design. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 233*(2), 451-464. https://doi.org/10.1177/0954406218778311

Cropley, D.H., Kaufman, J.C., & Cropley, A.J. (2011). Measuring creativity for innovation management. *Journal of Technology Management and Innovation, 6*(3), 13-30. http://doi.org/10.4067/S0718-27242011000300002

Cseh, G.M., & Jeffries, K.K. (2019). A scattered CAT: A critical evaluation of the Consensual Assessment Technique for creativity research. *Psychology of Aesthetics, Creativity, and*

the Arts, 13(2), 159-166. http://doi.org/10.1037/aca0000220

Dunbar, K. (1997). How scientists think: On-line creativity and conceptual change

in science. In T. B. Ward, S. M. Smith, & J. Vaid (Eds.), *Creative thought: An*

investigation of conceptual structures and processes (pp. 461-494). American

Psychological Association.

Dunn, T.J., Baguley, T., & Brundsen, V. (2014). From alpha to omega: A practical solution to

the pervasive problem of internal consistency estimation. *British Journal of Psychology,*

105(3), 399-412. https://doi.org/ 10.1111/bjop.12046

Eibel Process Co. v. Minnesota & Ontario Paper Co., 261 U.S. 45 (1923),

https://caselaw.findlaw.com/us-supreme-court/261/45.html

Feist Publications, Inc. v. Rural Telephone Service Company, Inc., 499 U.S. 340 (1991),

https://www.law.cornell.edu/supremecourt/text/499/340

Field, A. (2017). *Discovering statistics using IBM SPSS Statistics* (5th ed.). Sage.

Freeman, C., Son, J., & Roberts, L.B. (2015). Comparison of novice and expert evaluations of

apparel design illustrations using the Consensual Assessment Technique. *International*

Journal of Fashion Design, Technology and Education, 8(2), 122-130.

https://doi.org/10.1080/17543266.2015.1018960

Getzels, J., & Csikzentmihalyi, M. (1976). *The creative vision: A longitudinal study of problem-*

finding in art. Wiley.

Glass, R.L. (2006). *Software creativity 2.0.* Developer Books.

Good, J., Keenan, S.F., & Mishra, P. (2016). Education:= Coding + aesthetics; Aesthetic

understanding, computer science education, and computational thinking. *Journal of*

Computers in Mathematics and Science Teaching, 35(4), 313-318.

https://www.learntechlib.org/primary/p/174348/

Google LLC v. Oracle America, Inc., 141 S. Ct. 1183 (2021),

https://www.law.cornell.edu/supremecourt/text/18-956

Graham, J.M. (2006). Congeneric and (essentially) tau-equivalent estimates of score reliability.

Educational and Psychological Measurement, 66(6), 930-944. https://doi.org/

10.1177/0013164406288165

Graham v. John Deere Co., 383 U.S. 1 (1966),

https://caselaw.findlaw.com/us-supreme-court/383/1.html

Green, M., Seepersad, C.C., & Hölttä-Otto, K. (2014). Crowd-sourcing the evaluation of

creativity in conceptual design: A pilot study. *Proceedings of the ASME 2014

International Design Engineering Technical Conferences & Computers and Information

in Engineering Conference.* Volume 7: 2nd Biennial International Conference on

Dynamics for Design; 26th International Conference on Design Theory and

Methodology. V007T07A016. ASME. https://doi.org/10.1115/DETC2014-34434

Guilford, J.P. (1967). *The nature of human intelligence.* McGraw Hill.

Guttman, L. (1945). A basis for analyzing test-retest reliability. *Psychometrika, 10*(4), 255-282.

https://doi.org/10.1007/BF02288892

Jeffries, K.K. (2017). A CAT with caveats: Is the Consensual Assessment Technique a reliable

assessment measure of graphic design creativity? *International Journal of Design

Creativity and Innovation, 5*(1-2), 16-28.

https://doi.org/10.1080/21650349.2015.1084893

Katz-Buonincontro, J., Hass, R., & Perignat, E. (2020). Triangulating creativity: Examining

discrepancies across self-rated, quasi-expert-rated, and verbalized creativity in arts-based

learning. *The Journal of Creative Behavior, 54*(4), 948-963.

https://doi.org/10.1002/jocb.424

Kaufman, J.C., Baer, J., Cropley, D.H., Reiter-Palmon, R., & Sinnett, S. (2013). Furious

activity vs. understanding: How much expertise is needed to evaluate creative work?

Psychology of Aesthetics, Creativity, and the Arts, 7(4), 332–340.

https://doi.org/10.1037/a0034809

Kaufman, J.C., Beghetto, R.A., & Watson, C. (2016). Creative metacognition and self-ratings

of creative performance: A 4-C perspective. *Learning and Individual Differences, 51*,

394–399. https://doi.org/10.1016/j.lindif.2015.05.004

Kershaw, T.C., Bhowmick, S., Seepersad, C.C., & Hölttä-Otto, K. (2019). A decision tree based

methodology for evaluating creativity in engineering design. *Frontiers in Psychology, 10*,

Article 32. https://doi.org/10.3389/fpsyg.2019.00032

Knobelsdorf, M., & Romeike, R. (2008). Creativity as a pathway to computer science. In J.

Amillo, C. Laxer, E. Menasalvas, & A. Young (Eds.), *ITiCSE '08: Proceedings of the

13th Annual Conference on Innovation and Technology in Computer Science Education*

(pp. 286-290). Association for Computing Machinery.

https://doi.org/10.1145/1384271.1384347

Knuth, D.E. (1973). *Fundamental algorithms* (2nd ed.). Addison-Wesley.

Knuth, D.E. (1974). Computer programming as an art. *Communications of the ACM, 17*(12),

667-673. https://doi.org/10.1145/361604.361612

Kozbelt, A., Dexter, S., Dolese, M., & Ostrofsky, J. (2015). Regressive imagery in creative

problem-solving: Comparing verbal protocols of expert and novice visual artists and

computer programmers. *The Journal of Creative Behavior, 49*(4), 263-278.

https://doi.org/10.1002/jocb.64

Kozbelt, A., Dexter, S., Dolese, M., & Seidel, A. (2012). The aesthetics of software code: A

quantitative exploration. *Psychology of Creativity, Aesthetics, and the Arts, 6*(1), 57-65.

https://doi.org/10.1037/a0025426

Kudrowitz, B.M., & Wallace, D. (2013). Assessing the quality of ideas from prolific, early-stage

product ideation. *Journal of Engineering Design, 24*(2), 120-139.

http://dx.doi.org/10.1080/09544828.2012.676633

McDonald, R. P. (1999). *Test theory: A unified treatment*. Lawrence Erlbaum.

Moneta, G.B., Amabile, T.M., Schatzel, E.A., & Kramer, S.J. (2010). Multirater assessment of

creative contributions to team projects in organizations. *European Journal of Work and*

Organizational Psychology, 19(2), 150-176. https://doi.org/10.1080/13594320902815312

Nunnally, J.C., & Bernstein, I.H. (1994). *Psychometric theory* (3rd ed.). McGraw-Hill.

Patent Act of 1952, 35 U.S.C. § 1 *et seq.* (1952).

https://uscode.house.gov/view.xhtml?path=/prelim@title35&edition=prelim

Plucker, J.A.., Kaufman, J.C., Temple, J.S., & Qian, M. (2009). Do experts and novices evaluate

movies the same way? *Psychology & Marketing, 26*(5), 470-478.

https://doi.org/10.1002/mar.20283

Plucker, J.A., & Makel, M.C. (2010). Assessment of creativity. In J.C. Kaufman & R.J.

Sternberg (Eds.), *The Cambridge handbook of creativity* (pp. 48-73). Cambridge

University Press.

Rhodes, M. (1961). An analysis of creativity. *The Phi Delta Kappan, 42*(7), 305-310.

https://www.jstor.org/stable/20342603

Romeike, R. (2007). Applying creativity in CS high school education -- Criteria, teaching

example and evaluation. In R. Lister (Ed*.), Koli Calling '07: Proceedings of the Seventh*

Baltic Sea Conference on Computing Education Research (pp. 87–96). Australian

Computer Society, Inc.

Rosen, D.S., Kim, Y.E., Mirman, D., & Kounios, J. (2017). All you need to do is ask? The exhortation to be creative improves creative performance more for nonexpert than expert jazz musicians. *Psychology of Aesthetics, Creativity, and the Arts, 11*(4), 420-427. http://doi.org/10.1037/aca0000087

Runco, M.A., & Jaeger, G.J. (2012). The standard definition of creativity. *Creativity Research Journal, 24*(1), 92–96. https://doi.org/10.1080/10400419.2012.650092

Runco, M.A., McCarthy, K., & Svenson, E. (1994). Judgments of the creativity of artwork from students and professional artists. *The Journal of Psychology: Interdisciplinary and Applied, 128*(1), 23–31. https://doi.org/10.1080/00223980.1994.9712708

Sammet, J.E. & Hemmendinger, D. (2000). Programming languages. In A. Ralston, E.D. Reilly, & D. Hemmendinger (Eds.), *Encyclopedia of computer science* (4th ed., pp. 1470-1475). Wiley.

Satava v. Lowry, 323 F.3d 805 (9th Cir. 2003), https://casetext.com/case/satava-v-lowry?q=323%20F.3d%20805&PHONE_NUMBER_GROUP=P&sort=relevance&p=1&type=case

Saunders, D., & Thagard, P. (2005). Creativity in computer science. In J.C. Kaufman & J. Baer (Eds.), *Creativity across domains: Faces of the muse* (pp. 171-186). Psychology Press.

Sawyer, J. (2001). *When stuff happens*. The Sawyer Partnership.

Simonton, D.K. (1999). Creativity as blind variation and selective retention: Is the creative process Darwinian? *Psychological Inquiry, 10*(4), 309-328. https://doi.org/10.1207/S15327965PLI1004_4

Simonton, D.K. (2010). Creativity in highly eminent individuals. In J.C. Kaufman & R.J.

Sternberg (Eds.), *The Cambridge handbook of creativity* (pp. 174-188). Cambridge

University Press.

Sonnentag, S., Niessen, C. & Volmer, J. (2006). Expertise in software design. In K.A. Ericsson,

N. Charness, P.J. Feltovich, & R.R. Hoffman (Eds.), *The Cambridge handbook of

expertise and expert performance* (pp. 373-388). Cambridge University Press.

Sternberg, R.J., Kaufman, J.C., & Pretz, J.E. (2002). *The creativity conundrum: A propulsion

model of kinds of creative contributions*. Psychology Press.

Sternberg, R.J., & Lubart, T.I. (1999). The concept of creativity: Prospects and paradigms. In

R.J. Sternberg (Ed.), *Handbook of creativity* (pp. 3-15). Cambridge University Press.

Stokes, P.D. (1999). Learned variability levels: Implications for creativity. *Creativity Research

Journal, 12*(1), 37-45. https://doi.org/ 10.1207/s15326934crj1201_5

Stokes, P.D. (2001). Variations on Guilford's creative abilities. *Creativity Research Journal,

13*(3-4), 277-283. https://doi.org/ 10.1207/S15326934CRJ1334_05

Stokes, P.D. (2007). Using constraints to generate and sustain novelty. *Psychology of Aesthetics,

Creativity, and the Arts, 1*(2), 107-113. https://doi.org/ 10.1037/1931-3896.1.2.107

Storme, M., Myszkowski, N., Çelik, P., & Lubart, T. (2014). Learning to judge creativity: The

underlying mechanisms in creativity training for non-expert judges. *Learning and

Individual Differences, 32*, 19–25. https://doi.org/10.1016/j.lindif.2014.03.002

Tavakol, M., & Dennick, R. (2011). Making sense of Cronbach's alpha. *International Journal of

Medical Education, 2*, 53-55. https://doi.org/10.5116/ijme.4dfb.8dfd

Torrance, E.P. (1974). *Torrance Tests of Creative Thinking: Norms-technical manual.* Scholastic

Testing Service.

Watkins, M.W. (2017). The reliability of multidimensional neuropsychological measures: From

alpha to omega. *The Clinical Neuropsychologist, 31*(6-7), 1113-1126.

https://doi.org/10.1080/13854046.2017.1317364

Wieth, M.B., & Francis, A.P. (2018). Conflicts and consistencies in creativity research and

teaching. *Teaching of Psychology, 45*(4), 363-370.

https://doi.org/ 10.1177/0098628318796924

Xu, D., Wolz, U., Kumar, D., & Greenberg, I. (2018). Updating introductory computer science

with creative computation. In T. Barnes, D. Garcia, E.K. Hawthorne, & M.A. Pérez-

Quiñones (Eds.), *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on

Computer Science Education* (pp. 167-172). Association for Computing Machinery.

Young, M.E., & Racey, D. (2009). Judgments of creativity as a function of visual stimulus

variability. *Empirical Studies of the Arts, 27*(1), 89-107.

https://doi.org/10.2190/EM.27.1.e

Table 1

Program Control and Descriptive Variables (PCDV) Coding Scheme

Variable	Description
Subroutines	A count of the number of internal subroutines written by the programmer.
For Loops	A count of the number of "for" loops (using automated counters)
While Loops	A count of the number of "while" loops (using automated comparison)
If	A count of the number of "if" statements
Else	A count of the number of "else" statements
Case	A count of the number of "case" or "switch" statements
Go To	A count of the number of "go to" or "break" statements (causing a branch to occur)

Table 2

Demonstration of the PCDV for Different Program Samples

Program Sample	Subroutines	For Loops	While Loops	If	Else	Case	Go To	Resulting PCDV
A	00	02	00	01	01	00	00	00 02 00 01 01 00 00
B	01	00	01	04	03	00	02	01 00 01 04 03 00 02
C	02	00	00	02	00	01	00	02 00 00 02 00 01 00

Table 3

Summary of Factor Loadings for Oblimin Principal Axis Factor Analysis of the Modified

Creative Solutions Diagnosis Scale: Communalities, Eigenvalues, and Percentages of Variance

Item	1	Factor loading 2	3	4	5	Communality
Correctness			.75			.67
Performance			.75			.61
Appropriateness			.93			.82
Diagnosis				.59		.79
Prescription				.74		.82
Prognosis				.38	-.31	.51
Redirection					-.53	.70
Reinitiation					-.79	.78
Redefinition					-.69	.76
Generation	.31				-.57	.83
Convincingness		.65				.72
Pleasingness		.95				.90
Completeness		.83				.76
Gracefulness		.91				.82
Harmoniousness		.58				.57
Foundationality	.68					.78
Transferability	.85					.81
Germinality	.66					.74
Seminality	.88					.82
Vision	.83					.79
Pathfinding	.88					.74
Eigenvalue	10.29	3.02	1.24	.79	.41	
% of variance	48.99	14.36	5.91	3.74	1.96	
Factor correlations						
Factor 1	-					
Factor 2	.38	-				
Factor 3	.15	.44	-			
Factor 4	.46	.24	.11	-		
Factor 5	-.78	-.39	-.20	-.46	-	

Table 4

Differences between the Simple and Complex Problem on the PCDV

PCDV category	Simple Program	Complex Program	*t* (22)	*p*	*Cohen's d*
Subroutines	.61 (.50)	2.65 (2.72)	*t* = -3.69	.001	.770
For	.83 (.39)	8.87 (10.11)	*t* = -3.85	.001	.803
While	.17 (.39)	1.39 (.99)	*t* = -5.01	< .001	1.000
If	1.30 (.56)	5.96 (5.48)	*t* = -3.93	.001	.819
Else	.17 (.49)	1.13 (1.32)	*t* = -3.01	.006	.629
Go To	.71 (.55)	3.96 (3.56)	*t* = -4.47	< .001	.911

Note. PCDV = Program Control and Descriptive Variables. Descriptive data are in the form of M (SD). The PCDV category Case is not shown because it was not found in the data. Using the Bonferroni correction for multiple comparisons, any p-value smaller than .008 is statistically significant.

Table 5

Differences between the Simple and Complex Problem on the CSDS

CSDS category	Simple Program	Complex Program	*t* (22)	*p*	*Cohen's d*
Propulsion-Genesis	1.90 (.42)	2.44 (.36)	-4.89	< .001	1.000
Elegance	3.13 (.35)	3.43 (.45)	-2.38	.027	.496
Relevance and Effectiveness	3.64 (.24)	3.62 (.36)	.28	.784	.058
Problematization	2.34 (.45)	2.82 (.35)	-5.13	< .001	1.000

Note. CSDS = Creative Solution and Diagnosis Scale. Descriptive data are in the form of M(SD). Using the Bonferroni correction for multiple comparisons, any p-value smaller than .01 is statistically significant.

Appendix

Demonstration of the Program Control and Descriptive Variables (PCDV) Coding Scheme with Sample Programs

All of the following sample programs were designed to solve the problem *Compute the Hamming Distance Between Two Strings* from the Rosalind website (http://rosalind.info/problems/ba1g/). This problem is the simple problem referred to throughout the study.

As shown in Table A1, there are multiple ways to write a computer program to solve even simple problems. However, different approaches written in different languages can still lead to the same descriptive code, as demonstrated with the C-For and Perl programs (see Figure A1). These two programs—written in different programming languages—receive the same descriptive code using the PCDV: 00010002000000. The top program is written in C and has no *subroutines*, *while*, *else*, *case*, or *go to* statements, but contains two *if* statements and one *for loop*. The exact same is true for the Perl program below.

Likewise, writing a program in the same language can lead to different descriptive codes, as demonstrated with the C-While and C-Function programs (see Figure A2). These two programs—both written in the same programming language, C—received different descriptive codes, despite both solving the same algorithmic problem. The top program has no *subroutines*, *for*, *else*, *case*, or *go to* statements, but has two *if* statements and one *while* loop. The bottom program in Figure A2 has no *for*, *else*, *case*, or *go to* statements, but has one *subroutine*, two *if* statements and one *while* loop.

Table A1

Program Control and Descriptive Variables Analysis for Four Sample Programs

Program	Subroutines	For	While	If	Else	Case	Go To	Descriptive Code
Perl	00	01	00	02	00	00	00	00010002000000
C-For	00	01	00	02	00	00	00	00010002000000
C-While	00	00	01	02	00	00	00	00000102000000
C-Function	01	00	01	02	00	00	00	01000102000000

Figure A1

Illustration of the PCDV: Two Programs Written in Different Languages that Received the Same Descriptive Code

```perl
#!/usr/bin/perl -w
if ($#ARGV < 1) {
    print STDERR "2 inputs: seq1 and seq2\n";
    exit -1;
}
my $seq1 = $ARGV[0];
my $seq2 = $ARGV[1];
my $n = length ($seq1);
my $hamDist = 0;
for (my $i=0; $i <= $n; ++$i)
{
    if (substr($seq1, $i, 1) ne substr($seq2, $i, 1)) {
        $hamDist++;
    }
}
print "Hamming Distance is: $hamDist\n";
```

2 "if"   1 "for"

subs	for	while	if	else	case	go to	Program Control and Descriptive Variables (PCDV)
	1		2				**Count of PCDV**
00	**01**	00	**02**	00	00	00	**Perl Descriptive Code**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STR_LENGTH 1024
int main(int argc, char **argv)
{
  char *str1, *str2;
  int strLength, charIndex, hammingDistance = 0;
  if(argc != 3) {
    printf("USAGE: HD <str1> <str2>\n");
    return 0;
  }
  str1 = argv[1];
  str2 = argv[2];
  strLength = strlen(str1);
  for(charIndex = 0; charIndex < strLength; charIndex++)
  {
    if(*str1++ != *str2++)
      hammingDistance++;
  }
  printf("Distance = %d\n", hammingDistance);
  return 0;
}
```

2 "if"   1 "for"

subs	for	while	if	else	case	go to	Program Control and Descriptive Variables (PCDV)
	1		2				**Count of PCDV**
00	**01**	00	**02**	00	00	00	**C-For Descriptive Code**

Figure A2

Illustration of the PCDV: Two Programs Written in C that Received Different Descriptive Codes

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STR_LENGTH 1024
int main(int argc, char **argv)
{
  char *str1, *str2;
  int strLength, charIndex, hammingDistance = 0;
  if(argc != 3) {
    printf("USAGE: HD <str1> <str2>\n");
    return 0;
  }
  str1 = argv[1];
  str2 = argv[2];
  strLength = strlen(str1);
  while(charIndex < strLength)
  {
    if(*str1++ != *str2++)
      hammingDistance++;
    charIndex++;
  }
  printf("Distance = %d\n", hammingDistance);
  return 0;
}
```

2 "if"   1 "while"

subs	for	while	if	else	case	go to	Program Control and Descriptive Variables (PCDV)
		1	2				**Count of PCDV**
00	00	**01**	**02**	00	00	00	**C-While Descriptive Code**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STR_LENGTH 1024
int HammingDistance(char * str1, char *str2)
{
  int strLength, charIndex = 0, hammingDistance = 0;
  strLength = strlen(str1);
  while(charIndex < strLength)
  {
    if(*str1++ != *str2++)
      hammingDistance++;
    charIndex++;
  }
  return hammingDistance;
}
int main(int argc, char **argv)
{
  char *str1, *str2;
  if(argc != 3) {
    printf("USAGE: HD <str1> <str2>\n");
    return 0;
  }
  str1 = argv[1];
  str2 = argv[2];
  printf("Distance = %d\n", HammingDistance(str1, str2));
  return 0;
}
```

1 "subs"   1 "while"   2 "if"

subs	for	while	if	else	case	go to	Program Control and Descriptive Variables (PCDV)
1		1	2				**Count of PCDV**
01	00	**01**	**02**	00	00	00	**C-Function Descriptive Code**