Answering Question One in *Google v. Oracle*: The Creativity of Computer Programmers

Ralph D. Clifford, Firas Khatib, Trina C. Kershaw, and Adnan El-Nasan*

70 J. COPYRIGHT SOC'Y – (expected 2023)

Abstract

There is a misconception that computer programs are extremely limited by set expressions required by the computer system or the problem being coded and thus have little room for creativity. Under this fallacy, some argue that copyright protection for software is practically nonexistent as the *Feist* minimal creativity standard cannot be met. Others, including Google in the recent *Google v. Oracle* case before the Supreme Court, argue that even if the minimum creativity standard can be met, most aspects of software constitute ideas rather than expressions so, again, copyright protection fails under the merger doctrine.

Until recently, these factual assertions about the nature of computer programs and their creation have not been empirically tested. The authors have now done so. In a recently published, peer-reviewed study by the authors, the creativity leading to the writing of a computer program was established; indeed, the creativity used by a computer programmer is similar to that found in other disciplines that are acknowledged to be creative. The study took examples of computer programs written by multiple programmers

---

*. Ralph D. Clifford is a Professor of Law at the University of Massachusetts School of Law. He has a computer science degree and spent almost twenty years as a professional programmer before becoming a professor.

Firas Khatib is an Associate Professor of Computer and Information Science at the University of Massachusetts Dartmouth. He specializes in bioinformatics, citizen science, gamification, and applying these fields to K-12 education.

Trina C. Kershaw is a Professor of Psychology at the University of Massachusetts Dartmouth. One of her research specializations is measuring creativity in laboratory and applied science (i.e., engineering, computer programming, etc.) situations.

Adnan El-Nasan is a Full Time Lecturer of Computer and Information Science at the University of Massachusetts Dartmouth. His research interests include operating system optimization and security; cybersecurity, privacy, forensics and reverse engineering; and innovation and commercialization in emerging economies.

This article was supported by a writing grant from the University of Massachusetts School of Law. The authors wish to thank Jessica Almeida of the Law School Library who continues to be able to find the right materials based on the incomplete descriptions she is given.

to perform identical functions and applied recognized psychology-based tests to measure creativity. Although the study's programs were not particularly complex, the programmers found many significantly different and creative ways to code them. The study established that software—at least that more complicated than the program needed to print "Hello, world!"—vary greatly based on the creative expressions chosen by the program's author. This creative expression deserves full copyright protection.

Table of Contents

I. Introduction

In *Google LLC v. Oracle America Inc.,*[1] an appeal concerning the enforceability of Oracle's[2] Java API[3] software copyrights, the Supreme Court determined that the case could be resolved exclusively on a finding of fair use.[4] This left the first question posed in the case[5]—whether the computer code in question was even copyrightable—in limbo.[6] It

---

1. 141 S. Ct. 1183 (2021).

2. The software in question was developed by a predecessor corporation to Oracle, Sun Microsystems. *Id.* at 1190. Because all proprietary rights to this software are now owned by Oracle, Oracle will be called the author of the software in this article as the successor to Sun Microsystems.

3. "API" stands for **A**pplication **P**rogrammer **I**nterface. *See id.* at 1191. These allow programmers the ability to achieve commonly needed functions without having to write a program for each. *See id.*

The general concept of an API has existed by that name since at least the 1990s. *See* MARY SWEENEY, VISUAL BASIC FOR TESTERS 211 (2001) (discussing the "APIs" used in Microsoft Windows); Harold W. Thimbleby, *Java in* ENCYCLOPEDIA OF COMPUT. SCI. 937, 940 (Anthony Ralston *et al* eds., 4th ed. 2000) (describing APIs in Java). Of course, the concept without the name existed for decades before that. *See Macro Assemblers*, ENCYCLOPEDIA OF COMPUT. SCI. 99–100 (Anthony Ralston et al. ed. 4th ed. 2000) (describing achieving standard programming tasks by using the macro system available with 1960–1980-era IBM computers); IBM CORP., OS/VS-VM/370 ASSEMBLER PROGRAMMER'S GUIDE 69 (5th ed. 1982) (defining "library macro definition" as "IBM-supplied ... macro definitions").

In the *Google v. Oracle* case, the APIs that Oracle developed as part of its Java language were in litigation. *See generally*, Thimbleby, *supra*. For a current, comprehensive list of these APIs, see *Java Platform, Standard Edition & Java Development Kit Version 20 API Specification* (Draft 20-ea+1-3 ed. 2022), https://download.java.net/java/early_access/jdk20/docs/api/ [hereinafter *Java APIs*].

4. 141 S. Ct. at 1190 ("we assume, for argument's sake, that the material was copyrightable [b]ut we hold that the copying here at issue ... constituted a fair use.").

5. *See* Brief of the Petitioner at i, *Google LLC v. Oracle Am. Inc.*, 141 S. Ct. 1183 (2021) (No. 18-956), 2020 WL 104836 [hereinafter *Google's Brief*].

6. *See Google*, 141 S. Ct. at 1190; Adam Mossoff, *Declaring Computer Code Uncopyrightable with a Creative Fair Use Analysis*, 20 CATO SUP. CT. R. 237, 238–39

remains an important question, however, as the almost 400 billion dollar industry in annual market sales[7] is highly dependent on copyright to protect proprietary rights in software.[8]

As a practical matter, the lack of an answer to the first question by the Supreme Court leaves a critical gap in the legal foundation of software proprietary rights because so many challenge the appropriateness or comprehensiveness of copyrights for computer code.[9] Some of these question whether any computer software is creative enough to clear the *Feist* creativity requirement[10] while others may not directly question the existence of

---

(2020-21) . The lower appellate court had found that the APIs were protected by copyright. *See* Oracle Am., Inc. v. Google LLC, 886 F.3d 1179, 1210–11 (Fed. Cir. 2018), *rev'd on other grounds*, 141 S. Ct. 1183 (2021).

7. *See* Grand View Research, *Market Analysis Report* (Apr., 2021), https://www.grandviewresearch.com/industry-analysis/business-software-services-market.

8. *See, e.g.,* Rich Stim, *Copyrighting Your Software—Why Bother?*, https://fairuse.stanford.edu/overview/faqs/software/ (last visited Jan. 13, 2022) ("If you publish computer software, the single most important legal protection available to you is the federal copyright law.").

9. *See Google's Brief*, *supra* note 5, at 18 (arguing that the software APIs were uncopyrightable as "one of only a few possible means of expression" of the underlying idea of the APIs); Michael D. Murray, *Copyright, Originality, and the End of the* Scènes à Faire *and Merger Doctrines for Visual Works*, 58 BAYLOR L. REV. 779, 815 (2006) (accepting that "[t]echnical and practical requirements, design standards, and appropriate methods of operation dictate programming choices rather than the creative input of the creator...."); Aaron Kozbelt, Scott Dexter, Melissa Dolese, & Angelika Seidel, *The Aesthetics of Software Code: A Quantitative Exploration*, PSYCH. OF CREATIVITY, AESTHETICS, AND THE ARTS, Feb., 2012, at 57, 58 ("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.") [hereinafter Kozbelt (2012)].

10. *See* Michael D. Murray, *supra* note 9; Justin Hughes, *Restating Copyright Law's Originality Requirement*, 44 COLUM. J.L. & ARTS 383, 409 n.64 (2021) (noting that the courts have not defined creativity under *Feist*); Ralph D. Clifford, *Random Numbers, Chaos Theory and Cogitation: A Search for the Minimal Creativity Standard in Copyright Law*, 82 DENV. U. L. REV. 259, 268 (2004) (same). *See generally* Feist Pub., Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 345 (1991); 17 U.S.C. § 102(b) (2020); Baker v. Selden, 101 U.S. 99

underlying creativity, but assert that software can only be expressed in one way, triggering copyright unprotectability through the merger doctrine.[11] Of course, even where software is acknowledged to be copyrightable in theory, courts have mandated disqualifying tests for programs that prevent copyright protection in fact: the tests under the guise that an "efficient" piece of code is an idea rather than an expression, being the prime example.[12] Similarly, courts often engage in a *post hoc* analysis of the copyrightability of a program, ultimately rejecting protection because the code can no longer be considered creative because the market has so preferred that particular expression that it has become an industry standard.[13]

The fault for this does not rest solely with the judicial system not understanding the technology; instead, at least part of the confusion in all these cases stems from the lack

---

(1880).

11. *See Google's Brief*, *supra* note 5, at 18 (arguing that the software APIs were "one of only a few possible means of expression" of the underlying idea of the APIs). *See generally*, Michael D. Murray, *supra* note 9 (declaring that "[t]echnical and practical requirements, design standards, and appropriate methods of operation dictate programming choices rather than the creative input of the creator....").

12. Computer Assoc. Int'l, Inc. v. Altai, Inc., 982F.2d 693, 708 (2d Cir. 1992) ("While, hypothetically, there might be a myriad of ways in which a programmer may effectuate certain functions within a program,—i.e., express the idea embodied in a given subroutine—efficiency concerns may so narrow the practical range of choice as to make only one or two forms of expression workable options."). As is discussed in part IV *infra*, this conception of computer efficiency is not based on the reality of computer software works or on how programs are written.

13. *See* Jack E. Brown, *"Analytical Dissection" of Copyrighted Computer Software—Complicating the Simple and Confounding the Complex*, 25 ARIZ. ST. L.J. 801, 811–29 (1993) (discussing cases); *Google*, 141 S. Ct. at 1203 (allowing fair use copying because it "allow[ed] programmers to call upon those tasks without discarding a portion of a familiar programming language and learning a new one.").

of a scientific understanding of how computer programmers write code. Until recently, the basic question—"Are programmers creative when they write code?"—had not been answered empirically as only limited scientific studies of the discipline of coding had been done.[14] Unfortunately, these early foundational studies of programming are either based on a non-empirically-based assertion that such creativity is used to code,[15] or by determining that programmers feel that what they do is creative without subjecting these feelings to an empirical verification.[16]

Recently, the four authors completed the first empirical study that directly examines whether the coding process is creative.[17] As is more fully described in the next section, the short answer is "Yes, programmers are creative when they write code." In addition, both the merger doctrine and the efficiency-equals-idea limitations asserted against computer

---

14. *See* P.J. Barnett & R. Romeike, *Creativity Within Computer Science in* CAMBRIDGE HANDBOOK OF CREATIVITY ACROSS DOMAINS 299 (J.C. Kaufman, V.P. Glăveanu, & J. Baer eds. 2017); R.L. GLASS, SOFTWARE CREATIVITY 2.0 (2006); M. Knobelsdorf, & R. Romeike, *Creativity as a Pathway to Computer Science, in* ITiCSE '08: PROC. OF THE 13TH ANN. CONF. ON INNOVATION AND TECH. IN COMPUT. SCI. EDUC. 286 (J. Amillo, C. Laxer, E. Menasalvas, & A. Young eds., 2008); Kozbelt (2012), *supra* note 9; Aaron Kozbelt, Scott Dexter, Melissa Dolese, Daniel Meredith & Justin Ostrofsky, *Regressive Imagery in Creative Problem-Solving: Comparing Verbal Protocols of Expert and Novice Visual Artists and Computer Programmers*, 49 J. OF CREATIVE BEHAV. Dec., 2015, at 263 [hereinafter Kozbelt (2015)]; D. Saunders & P. Thagard, *Creativity in Computer Science in* CREATIVITY ACROSS DOMAINS: FACES OF THE MUSE 171 (J.C. Kaufman & J. Baer eds. 2005).

15. *See* DONALD E. KNUTH, FUNDAMENTAL ALGORITHMS at v (2d ed. 1973); Donald E. Knuth, *Computer Programming as an Art*, 17 COMM. OF THE ACM 667 (1974).

16. *See* Kozbelt (2012), *supra* note 9; Kozbelt (2015), *supra* note 14.

17. Trina C. Kershaw, Ralph D. Clifford, Firas Khatib, & Adnan El-Nasan, *An Initial Examination of Computer Programs as Creative Works*, PSYCH. OF AESTHETICS, CREATIVITY, AND THE ARTS (Jan. 27, 2022), http://dx.doi.org/10.1037/aca0000457 (peer-reviewed) [hereinafter *Programming Creativity*].

software copyrights are likely to be of limited usefulness as the reality of how programs are created demonstrate the factual inapplicability of both. After these study findings are summarized, the final section of this article will discuss the appropriate—and factually-based—contours of these copyright doctrines as they are applied to computer software. Through the discussion, it will become clear that the answer to question one in *Google v. Oracle* should have been "Yes, computer software is a type of creatively-based expression of ideas that obtains copyright protection, including the APIs in question in the case."[18]

II. A Description of the Empirical Examination of Computer Programs that Found Them to be Creative Products

To investigate whether computer programmers are creative when they write software, the authors designed an empirical study.[19] Two scales were used to assess creativity.[20] The first scale, known as the "Creative Solution Diagnosis Scale" or "CSDS" is an existing, well-regarded psychological measurement of the human creativity involved in creating functional products such as engineering designs and computer software.[21] It

---

18. This article does not examine the appropriateness of the fair use finding in *Google*. This is consistent with the authors' position in their *amicus* brief in the case. *See* Brief of *Amicus Curiae* Interdisciplinary Research Team on Programmer Creativity in Support of Respondent at 4, *Google LLC v. Oracle Am. Inc.*, 141 S. Ct. 1183 (2021) (No. 18-956).

19. *See Programming Creativity, supra* note 17. This law review article will summarize the methodology and findings of the peer-reviewed *Programming Creativity* study, but will not present the full details of the empirical approach used nor of the statistical analysis done. For a more comprehensive understanding of these aspects of the study including a comprehensive presentation of the statistical methods used, please consult the scientific work. *See id*.

20. *See id.* at 3.

21. *See id.* at 2. The CSDS was designed to measure the creativity involved in building functional products. *See* David H. Cropley & James C. Kaufman, *Measuring Functional Creativity: Non-Expert Raters and the Creative Solution Diagnosis Scale*, J. OF CREATIVE

evaluates creativity using a defined set of factors that have been found to be associated with different aspects of creativity.[22] To use the CSDS, the product is examined by individuals who have the expertise to understand it and how it works.[23] For the *Programming Creativity* study, the software examples were appraised by expert-, peer-, and self-evaluators.[24]

As a supplement to the CSDS and to provide a more objective measurement of the expressive variation within the code, the authors defined a second scale known as the "Program Control and Descriptive Variables" or "PCDV" evaluation.[25] This scale involved the analysis of the source code produced in the study to determine the number of times each fundamental control statement (such as "IF" or "FOR") was used.[26] Each program was examined by our research assistants (who were trained in computer science, not law) who counted the number of each kind of control statement it contained.[27] By combining the seven counts into a single descriptive numeric code, a fourteen-digit number was created

---

BEHAVIOR, June, 2012, at 119, 120–22.

22. *See Programming Creativity, supra* note 17, at 4.

23. *See* Cropley & Kaufman, *supra* note 21, at 123–26.

24. *See Programming Creativity, supra* note 17, at 4. The expert evaluator was Professor El-Nasan; the students evaluated their own program as well as programs written by other students. *Id.*

25. *See id.* at 4–5.

26. *Id.* The statements chosen for analysis are universal as all computer languages capable of implementing a defined procedure contain them, either with its common name, i.e., "FOR" statement, or with a variant, i.e., "DO" or "PERFORM." For examples of the PCDV in operation, see *id.* at 12–13.

27. *See id.* at 5.

that captured the essence of the expression of the algorithm being implemented while excluding trivial variations such as differences among them in the names chosen for variables.[28] If two independently developed computer programs have the same code, the PCDV supports the conclusion that there is no meaningful variation in expression between the two programs.[29] If, on the other hand, the PCDV code is different, a nontrivial variation exists.[30] To determine the overall variation within a set of programs, the total number of unique PCDV codes can be divided by the total number of programs, giving a single number (the "PCDV Ratio") that establishes the percentage of variation among the code samples.[31]

---

28. *See id.* This is not to imply that the choice of variable names is irrelevant to measuring creativity as choosing variable names "that mean something" has long been recommended as part of good programming practice. BRIAN W. KERNIGHAN & P.J. PLAUGER, THE ELEMENTS OF PROGRAMMING STYLE 145 (2d ed. 1978). Of course, what "means something" to one person might be undecipherable by another. *See* Alvaro Videla, *Meaning and Context in Computer Programs*, COMM. OF THE ACM, May, 2022, at 56. Consequently, the choice of a variable name is likely to be creative, too. After all, Hemingway could have named his novel "*The Bull Fight*," but doing so could easily change the nature of the work.

As the *Programming Creativity* study was done, no achievable way to measure the difference between variable names objectively was found. If it was simply added to the PCDV statistic, all programs would be found to be unique. At the detail level, however, one needs to decide if calling a variable *i* is different and creative from calling it *j* or *n*. Consequently, the decision was made to not measure this variation among the program samples submitted as this lessens the quantity of variation that would be found.

29. *See Programming Creativity, supra* note 17*, at 5.*

30. *See id.*

31. *See id.* A small number would indicate little or no variation (the smallest possible number is the reciprocal of the sample size); a large number indicates more variation (the largest possible number is one which indicates every example program is unique). *See id.*

We used graduate-level students, all of whom were experienced programmers, to obtain the data used in the study.[32] Each student, as part of their homework assignment in the course being taught by Professor Khatib, was required to create several programs that implemented bioinformatics algorithms.[33] The assignments used in the study required the students to solve two coding problems both correctly and using no more than five minutes of execution time.[34] Other than these minimal instructions, each student was free to choose how to solve the problem.[35]

Twenty-three students produced code that solved the two problems.[36] Each of these programs were evaluated using both the PCDV and CSDS scales.[37] The primary findings of the study are presented in the next two sections. As the PCDV's analysis of expressive

---

32. *See id.* at 3–4.

33. *See id.* at 4.

34. *See Id.* The simpler problem used in the study computed the Hamming distance between two strings. *Id.* This algorithm is useful in studying mutations in DNA. *Id.* The more complex problem used was to reconstruct a string from its k-mer composition. *Id.* "This algorithm is important for genome assembly where long strands of DNA have been fragmented into shorter pieces (k-mers)." *Id.*

The five minute limit of time was significant as the quantity of data being processed by the students was large. *Id.* It was impossible for a student to solve the problem without using a computer program. *Id.*

35. *See id.* This freedom included the choice of programming language to use.

36. *See id.* Several students could not produce functional code for the two problems. *Id.* at 3–4. As the students were all graduate students with significant exposure to programming, this serves as an indication of the nontrivial nature of the coding required. *See id.*

37. *Id.* at 4–5.

variations is a critical predecessor to finding creativity, it will be discussed first, followed by the CSDS.

A. Computer Programs—Even Simple Ones—Result from Numerous Expressive Choices Made by the Programmer (The PCDV)

The amount of variation found in the *Programming Creativity* study was surprising, particularly for the simpler of the two programs. For the simpler problem, the calculated PCDV Ratio was 0.478 indicating that almost half of the students submitted code that differed from the programs submitted by others.[38] For the more complex problem the PCDV Ratio was significantly higher at 0.870 establishing that only a few students submitted code that was expressively similar to another student's program while most created software that was measurably different from anyone else's code.[39]

Based on this, the conclusion is that programmers exercise considerable variation in the way they write programs. As the *Programming Creativity* study concluded,

> 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.[40]

38. *Id.* at 7.

39. *Id.*

40. *Id.*

Of course, variation alone does not establish creativity although it is certainly a precursor for it.[41] This is why the study included a second evaluation scale, the CSDS, which is designed to measure whether creativity itself is demonstrated in the programs being examined.

B. Computer Programs—Even Simple Ones—Demonstrate Creativity (The CSDS)

The purpose of the CSDS is to measure whether human creativity is present in a product.[42]

> [The 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, but it also captures the aesthetic components of the product, which are important for evaluation of creativity in multiple domains, including computer programming.[43]

"Relevance and effectiveness" means that the "[program] displays knowledge of existing facts and principles and satisfies the requirement in the problem statement."[44] Both "problematization" and "propulsion" measure aspects of the program's novelty.[45] "Problematization" determines whether the program "draws attention to problems in what already exists," while "Propulsion" evaluates whether the program "adds to existing knowledge,"

---

41. *See id.* at 2.

42. *See id;* Cropley & Kaufman, *supra* note 21, at 123–26.

43. *Programming Creativity, supra* note 17, at 2.

44. Cropley & Kaufman, *supra* note 21, at 124 (Table 2, Line 1).

45. *Compare id.* at 124 (Table 2, line 2, column "indicators") *with id.* at 133 (Table 7, columns "Problematization" and "Propulsion").

and whether it "develops new knowledge."[46] Elegance addresses whether the "[program] strikes observers as beautiful (external elegance) [and] is well worked out and hangs together (internal elegance)."[47] Finally, "genesis" measures whether the program contains "ideas [that] go beyond the immediate situation."[48]

In the *Programming Creativity* study, each CSDS factor was measured for the collected computer software.[49] Relevance and effectiveness—did the example computer program solve the problem it was designed to implement and did it work within the constraints allowed?—were mostly confirmed automatically because the students could not submit their code unless it generated the correct answer within the allowed time-frame.[50] To determine relevancy and effectiveness beyond the minimum, expert-, peer-, and self-evaluations were collected.[51] Similarly for novelty, some of the analysis was easily completed as the problem to be solved was fully defined in the homework assignment which minimized the importance of problematization. That reduced the question of novelty to a simpler inquiry—did the code contain anything that the evaluators found to be unexpected or unique?[52] In the study, this component of novelty was measured using

---

46. *Id.* at 124 (Table 2, Line 2); *id.* at 133 (Table 7, columns "Problematization" and "Propulsion").

47. *Id.* at 125 (Table 2, line 3).

48. *Id.* (Table 2, Line 4).

49. *See Programming Creativity, supra* note 17, at 6.

50. *See id.* at 4 (describing the Rosalind website).

51. *See id.* at 3–4.

52. Indeed, for statistical reasons, the factor analysis performed merged the propulsion and genesis factors into one. *See id.* at 6 ("Due to the multicollinearity between these

expert-, peer-, and self-evaluation of each program.[53] Evaluating elegance within each code sample was nothing less than an examination of Professor Knuth's "art" factor[54]—did the code look special either superficially or in its internal workings?[55] Again, the study used a multi-rater analysis protocol to evaluate this.[56] Finally, genesis-propulsion required an analysis of whether the programmer engaged in broader problem-solving—did the programmer go beyond the bare minimum needed to satisfy the homework assignment? This scale, too, used a multi-rater analysis protocol.[57]

The study included consistency confirmations as the evaluations needed by the CSDS were subjective. First, raters who submitted facially defective evaluations (by rating every programmer "excellent" in every evaluation point, for example) were removed from the data.[58] Then, the evaluation of the remaining participants were shown to be highly consistent with each other using the statistical methods that are designed to confirm this.[59]

The CSDS findings of the *Programmer Creativity* study are clear. The study's hypothesis posited that if creativity is a significant factor in the development of a computer

---

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.").

53. *Id.*

54. *See supra* authorities cited note 15.

55. *Programming Creativity, supra* note 17, at 4

56. *Id.*

57. *Id.*

58. *See id.* at 5.

59. *See id.* at 5–6 (discussing results of Cronbach's alpha, Guttman's lambda, and McDonald's omega).

program, the CSDS rankings would show this directly[60] and it would be expected that a more complex problem would be more highly rated on the CSDS problematization and propulsion-genesis scales than a simple one.[61] The study confirmed both findings.[62]

C. Study Conclusion: Nontrivial Computer Programs Are Creative Expressions

There is significant expressive creativity used to express a computer program, even simple ones like the easier problem used in *Programmer Creativity*. The PCDV Ratio of 0.478 demonstrated that there were a multitude of ways to express even a somewhat simple program.[63] As code becomes more complex, the possible variants grow quickly as demonstrated by the PCDV Ratio of 0.870 on the more complex program. "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."[64] Further,

---

60. *See id.* at 2.

61. *Id.* at 3 & 7 (discussing second principle hypothesis). An analogy to this would be to compare a "normal" highway bridge with one that crosses a major waterway. Most bridges are fairly mundane and would not be described by most as creative, but some gain fame and are considered beautiful and creative, e.g., the Golden Gate or George Washington bridges, partially because of the complexity of the engineering problem that has to be solved. Similarly, a more complicated computer program is more likely to allow its programmer to achieve something that is special.

62. *Id.* at 7 & Table 5.

63. *Id.* at 7.

64. *Id.* at 7–8.

as problems get more complex, more creativity is expressed.[65] To apply this in the *Google* case, telling multiple programmers to write the code needed to define the APIs for Java would result in a PCDV Ratio near the maximum of one because of the complexity of this task.

When the CSDS is combined with the PCDV, the conclusion that programmers are creative is inescapable. Unfortunately, as the next part of this article explores, the legal system has failed to appreciate this.

III. The Legal Consequences that Result from the Finding that Programmers Express Creativity

A. The *Feist* Creativity Requirement in Copyright Law is Easily Satisfied for the Vast Majority of Computer Software, Including the APIs in *Google v. Oracle*

The Supreme Court has defined a minimal qualification for any expression to be protected by copyright. To be copyrightable, an expression must

> possess[] at least some minimal degree of creativity. To be sure, the requisite level of creativity is extremely low; even a slight amount will suffice. The vast majority of works make the grade quite easily, as they possess some creative spark, "no matter how crude, humble or obvious" it might be. Originality does not signify novelty; a work may be original even though it closely resembles other works so long as the similarity is fortuitous, not the result of copying.[66]

65. *Id.* at 7 (finding that both variation as measured by the PCDV Ratio and creativity factors as measured by the CSDS increase with complexity).

66. Feist Publ's, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 345 (1991).

As the Court held, the requisite established by this standard is an "extremely low" one; indeed, if multiple expressions of an idea are possible, the choice of one over the other constitutes an exercise of creativity.[67]

When applied to computer software, the *Feist* test is easily cleared. As the authors' *Programming Creativity* study established, programmers make numerous choices as they write programs.[68] Most of these choices of expression are the programmer's own and are not dictated by external factors or supposed efficiency considerations.[69] Even the Supreme Court seemed to acknowledge that multiple versions of software was possible, including the Java APIs in litigation.[70] Indeed, to appropriate Oracle's APIs, Google "copied roughly 11,500 lines of code from the Java SE program."[71] The programs studied in *Programming Creativity* were radically shorter and consequently more limited in possible variation than Oracle's API code. The simpler programs involved in the study had as few as six, and no more than twenty-six lines of code.[72] Despite the minuscule size of these programs created

---

67. *See id.* at 348; Clifford, *supra* note 10, at 295–96; *Programming Creativity*, *supra* note 17, at 2; *supra* section II.A.

68. *See Programming Creativity*, *supra* note 17, at 7 ("[The study] result[s] demonstrates that there is a large variation of programming expressions that can be used to solve even simple coding problems."); *supra* section II.A.

69. *See Programming Creativity*, *supra* note 17, at 3 (imposing an identical execution speed maximum requirement on all programmers); *infra* section IV (discussing the fallacy of using computer efficiency in copyright analysis). *See also* authorities cited, *supra* note 14.

70. *See Google*, 141 S. Ct. at 1194.

71. *Id.* at 1191.

72. *See* files in the directory *"Spring 2019 data\homework info\HWK3 Prob 2-2 code"* on file with the authors.

within the study, eleven unique solutions were created for them.[73] With 11,500 lines of code, the possible variations from which to choose in creating this system would have been massively larger than the choices available in the authors' study. The direct conclusion of this is that the APIs in litigation in the *Google* case demonstrated far more creativity in their creation than is required by *Feist.*

Expanding from the *Oracle v. Google* case, any argument that asserts that a nontrivial computer program lacks the modicum of creativity demanded by *Feist*[74] should be treated with disdain. Writing a program, even one as short as one with ten lines of code, involves numerous creative choices that are sufficient to clear the "extremely low"[75] copyright creativity requirement.[76] Any program involved in litigation will most certainly be longer than this.

It should be noted that the number of lines of code is not a very precise calculation. Among the expressive tools available to a programmer is how the lines of code are entered. For most modern languages, a single computer command can be entered across multiple lines. By breaking the command into multiple lines, and typically indenting some parts of it, the programmer can make the code more easily understood by a human reader without impacting the operation of the program. *See* KERNIGHAN & PLAUGER, *supra* note 28, at 1–3 & 146–50. Similarly, programmers are encouraged to include "comments" within their code to explain what the code does. *See id.* at 141–45. As these comments are generally not considered to be "lines of code," they have been omitted from the count given. In reality, what is being counted is the number of commands used, not the physical number of lines.

73. *Programming Creativity*, *supra* note 17, at 7.

74. *Feist*, 499 U.S. at 346.

75. *Id.* at 345.

76. *See* Clifford, *supra* note 10, at 295–96.

B. As with Other Creative Expressions, the Merger Doctrine Needs to be Limited to its Proper, Narrow Role

Just because a computer program is the result of creative expression does not mean that all aspects of the software are protected under the Copyright Act. Congress, in section 102(b),[77] codified the long-recognized dichotomy between expressions that are within the ambit of copyright and the underlying ideas that are not.[78] Of course, as has been recognized for almost as long, splitting an idea from its expression is not a trivial undertaking;[79] indeed, this process has been particularly challenging for computer software.[80] Unfortunately, the misconceptions about computer software being addressed in this article and by the authors' *Programming Creativity* study too often form the basis of the difficulty.[81] If the courts—often based on the expressed views of commentators[82]—fail

---

77. 17 U.S.C. § 102(b) (2020).

78. H. REP. NO. 94-1476, at 57 (1976), *as reprinted in* 1976 U.S.C.C.A.N. 5659, 5670 ("Section 102(b) in no way enlarges or contracts the scope of copyright protection under the present law."). The basic statement of the idea-expression dichotomy can be found in *Baker v. Selden*, 101 U.S. (11 Otto) 99 (1879), and the hundreds of cases that have interpreted it.

79. *See, e.g.*, Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (1930) (L. Hand, C.J.).

80. *E.g.*, Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823, 836 (10th Cir. 1993) ("Distinguishing between ideas and the expression of those ideas is not an easy endeavor...."). *Compare* Computer Assoc. Int'l, Inc. v. Altai, Inc., 982F.2d 693, 703–06 (2d Cir. 1992) *with* Whelan Assocs., Inc. v. Jaslow Dental Lab'y, Inc., 797 F.2d 1222, 1235–37 (3d Cir. 1986). *See generally*, Clifford, *supra* note 10, at 282–89.

81. *See, e.g., Computer Assoc.* at 707–10.

82. The copyright expertise of both Melville Nimmer (original author of NIMMER ON COPYRIGHT) and David Nimmer (current author) is appropriately and widely recognized. *See, e.g.*,MacLean Assocs., Inc. v. Wm. M. Mercer-Meidinger-Hansen, Inc., 952 F.2d 769, 778 (3d Cir. 1991); Galiano v. Harrah's Operating Co., 416 F.3d 411, 419 (5th Cir. 2005);

to understand the nature of computer software and its creation, the resulting decisions become problematic.

Bateman v. Mnemonics, Inc.[83] provides a good example of this. The court, relying on the *Computer Assoc. v. Altai* case for key factual conclusions (which in turn had quoted NIMMER ON COPYRIGHT), stated that "in many instances it is virtually impossible to write a program to perform particular functions in a specific computing environment without employing standard techniques."[84] In reality, the opposite is true.[85] Given the same task and programming environment, the programmers in the authors' study produced widely divergent code.[86] The simpler coding task showed a variation in expression of almost 50% with eleven unique solutions from twenty-three programmers while the more complicated

---

VMG Salsoul, LLC v. Ciccone, 824 F.3d 871, 880 (9th Cir. 2016). Neither, however, have computer science training or experience. *See David Nimmer*, IRELL & MANELLA LLP (last visited June 12, 2022, 10:15 AM), https://www.irell.com/professionals-david-nimmer. Unfortunately, the factual misconception disproved by *Programming Creativity* is stated as a definitive fact in NIMMER ON COPYRIGHT, 4 DAVID NIMMER, NIMMER OF COPYRIGHT § 13.03[F][3] (2022) ("it is virtually impossible to write a program to perform particular functions in a specific computing environment without employing standard techniques."); indeed, this provision has been quoted in the case law to establish that proposition. *See, e.g.,* Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 709 (2d Cir. 1992).

When the authorities cited in NIMMER ON COPYRIGHT for this factual conclusion are examined, they turn out to be exclusively legal authorities. DAVID NIMMER at § 13.03[F][3], n.312. Of course, the legal authorities cited in NIMMER ON COPYRIGHT typically use that authority for so ruling. Consequently, these citations establish a circle of authorities as many of the cases cited in the footnote are authorities that relied on—indeed, often quoted—the misstatement in NIMMER. It is time for this to be corrected.

83. 79 F.3d 1532 (11th Cir. 1996).

84. *Id.* at 709 (quoting *Computer Associates v. Altai* which in turn was quoting NIMMER ON COPYRIGHT). *See supra* note 82 (establishing the circular nature of this authority).

85. *Supra* section II.A.

86. *See id.*

software had a variation of about 85% with twenty unique solutions from the twenty-three programmers.[87] Asserting that it is "impossible" to write code to accomplish the same thing in different ways is demonstrably wrong; in fact, for nontrivial programs, expressive differences will almost always happen.

Unfortunately, once the factual misstatement is accepted and widely circulated, a compounding misapplication of the copyright merger doctrine can occur.[88] The doctrine operates when

> it is so difficult to distinguish between an idea and its expression that the two are said to merge. Thus, when there is essentially *only one way to express an idea*, copying the expression will not be barred, since protecting the expression in such circumstances would confer a monopoly of the idea upon the copyright owner free of the conditions and limitations imposed by the patent law. By denying protection to an expression that is merged with its underlying idea, we prevent an author from monopolizing an idea merely by copyrighting a few expressions of it.[89]

But as the authors' *Programming Creativity* study establishes—other than for unrealistically simplistic "print 'Hello, World'"-type programs—there are always a multitude of different expressions possible to implement any computer program.[90] Consequently, each new programmer can re-express the underlying programming ideas (because this is always possible as a practical matter) and, if the coding is done without violating the first

---

87. *Programming Creativity*, *supra* note 17, at 7.

88. *See, e.g.,* Herbert Rosenthal Jewelry Corp. v. Kalpakian, 446 F.2d 738, 742 (9th Cir. 1971) ("When the idea and its expression are thus inseparable, copying the expression will not be barred...." (quotation marks omitted)).

89. Mason v. Montgomery Data, Inc., 967 F.2d 135, 138 (5th Cir. 1992) (emphasis added) (quotation marks and citations omitted).

90. *Programming Creativity*, *supra* note 17, at 7.

programmers rights against copying and derivation under of the Copyright Act, the new programmer's code will not be infringing.[91] This would be true even in the unlikely event that the second programmer produces identical code.[92] In summary, as with most other copyrighted works, the merger doctrine is the rare exception for computer programs so courts should hesitate before applying it.[93]

Of course, Google attempted to raise merger as a reason not to allow copyright remedies in the *Google v. Oracle* case.[94] Specifically, Google asserted that the appropriated code had to be copied "[b]ecause no other instructions can perform the declarations'

---

91. *See* 17 U.S.C. §§ 106(1) & 106(2) (2020) (establishing the rights against copying and derivation).

92. *See* Mazer v. Stein, 347 U.S. 201, 217–18 (1954) (holding that two people may independently create identical works which would entitle both of them to copyrights). Some courts fail to limit proof of substantial similarity to its proper role in copyright litigation, triggering confusion about this possibility. *See, e.g.*, Universal Athletic Sales Co. v. Salkeld, 511 F.2d 904, 907 (3d Cir. 1975) (requiring proof of substantial similarity to prove infringement). More appropriately, proof of substantial similarity, when combined with access to the first author's work, raises a presumption that copying occurred. *See* Keeler Brass Co. v. Cont'l Brass Co., 862 F.2d 1063, 1065 (4th Cir. 1988) ("As most courts have recognized, persuasive direct evidence of copying is seldom available to a plaintiff in an infringement controversy. For that reason, courts have generally accepted circumstantial evidence to create a presumption of copying. To raise this presumption, the plaintiff must show that the alleged copier had access to the material and that the original material and the alleged copy are substantially similar."). The difference is significant as presumptive proof can be rebutted by evidence that contradicts the ultimate conclusion that copying occurred. *See id.* at 1066. Without this rebuttal being possible, *Mazer's* independent creation would be impossible.

93. *See, e.g.*, Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823, 838 (10th Cir. 1993) ("The merger doctrine is applied as a prophylactic device to ensure that courts do not unwittingly grant protection to an idea by granting exclusive rights to the only, or one of only a few, means of expressing that idea.").

94. *See Google's Brief*, *supra* note 5, at 14.

function, [so] merger excludes them from copyright protection."[95] To examine this assertion both the nature of the code taken and the nature of the function it performed must be examined.

1. Computer Software that Performs a Nontrivial Function Does Not Merge

The software involved in the case was far from simple or trivial. All told, Google appropriated approximately 11,500 lines of code.[96] These lines of code defined the APIs that create a set of pre-written functions that a programmer can use without having to write the needed operative code.[97] Some of these routines are simple like the "maximum" API discussed by the Supreme Court[98] but most APIs are far more complicated, allowing a programmer to create a new window on the screen, sort data into order, retrieve data from a database, or implement security protocols as examples.[99]

It is important to note that the lines of code taken were not just an alphabetical list of available APIs; instead, they were an organized list where Oracle had placed each API

---

95. *Id.*

96. *Google*, 141 S. Ct. at 1191. In comparison, the simpler program in the authors' *Programming Creativity* study had tens of lines of code. *See supra* text accompanying note 72.

97. *See Google* at 1191. Google only copied the definitional code, not the operational code, *see id.*, so its copying was not comprehensive—a vast majority of the needed code to reproduce the APIs was independently created. *See supra* note 92. *See generally supra* note 3.

98. *See Google* at 1192. It must be noted that there are 154 different "maximum" functions defined within the Oracle APIs, each achieving a different processing task. *See Java APIs, supra* note 3 (searching for "maximum").

99. *See* Thimbleby, *supra* note 3, at 940; *Java APIs, supra* note 3.

into a deliberately created structure that it felt would make each API easier to find by the programmer, thus making its Java language easier to use.[100] This structure, being the creative expression of Oracle, is also within the ambit of copyright protection as databases can be protected by copyright based on the creativity used to select items for inclusion and for the overall organization imposed on them.[101] This expressive aspect of the Oracle APIs was copied by Google, also.

Consequently, for Google to argue that merger applies to the APIs is nonsense. With over 2,000 APIs defined, all of which would operate just as effectively were they to be placed into a different methods, classes, and packages, Oracle's arrangement is patently not the only choice available. There could also be substantial variations in the names of the actual APIs. The "maximum" API discussed by the Supreme Court[102] could have just as easily been named "max," "larger," "bigger," or anything else that captures a comparison of relative size. While on a single API, this variation would likely be determined to be *de minimus*,[103] when applied over the thousands of APIs defined within Java each of

---

100. *See Google* at 1191 ("[E]ach individual task is known as a 'method.' The API groups somewhat similar methods into larger 'classes,' and groups somewhat similar classes into larger 'packages.'").

101. *See* Feist Publ'ns, Inc. v. Rural Tel. Serv. Co., Inc., 499 U.S. 340, 348–49 (1991); Infogroup, Inc. v. Database LLC, 956 F.3d 1063, 1066 (8th Cir. 2020) (finding database creatively selected and organized).

102. *See Google* at 1192.

103. *See, e.g.,* Sandoval v. New Line Cinema Corp., 147 F.3d 215, 217 (2d Cir. 1998) (defining "*de minimus*" by holding that "the alleged infringer must demonstrate that the copying of the protected material is so trivial as to fall below the quantitative threshold of [copyright].") (quotation marks and citation removed).

which could similarly be renamed, the variation becomes significant enough for copyright protection.

Of course, Google argued that these APIs are not "normal" software because they are part of the Java language.[104] As the next section discusses, however, this does not make a difference.[105]

2. Software that Creates a Programming Language Does Not Merge

It must be noted that Java is not a natural thing.[106] Before Oracle's employees expressed the programming language in the mid 1990s,[107] there was nothing known as Java. Further, Java is expressed as a computer program[108] so, because it was intended "to be used directly ... in a computer in order to bring about a certain result," copyright protection was expressly intended by Congress.[109]

Despite this, it seems that Google considers the Java language itself as a fact.[110] The reality is different, however, as programming languages, including Java, are brought

---

104. *See Google's Brief*, *supra* note 5, at 20.

105. Again, a fair use analysis is not being done. *See supra* note 18.

106. *Cf.* Castle Rock Ent., Inc. v. Carol Pub. Grp., Inc., 150 F.3d 132, 138–39 (2d Cir. 1998) ("each [*Seinfeld* television show] trivia question is based directly upon original, protectable expression in *Seinfeld*. As noted by the district court, The [defendant] did not copy ... unprotected facts, but, rather, creative expression.").

107. Thimbleby, *supra* note 3, at 937.

108. *See id.* at 937 & 940.

109. 17 U.S.C. § 101 (2020) (defining "computer program").

110. *See Google's Brief*, *supra* note 5, at 3–4 & 19–20.

into existence when their authors/programmers creatively express them.[111] In Java's case, its author first fixed it in 1995.[112] In copyright parlance, therefore, Java computer program is the "work of authorship" that has been fixed and is owned by Oracle.[113] For merger to apply to Java, it would have to be unique, not in comparison to itself as all works of authorship are identical to themselves, but in comparison to the environment of similar expressions. In other words, can an author express additional computer languages—or even more strictly an object-oriented computer language—or has Oracle monopolized the field with Java.[114] The facts clearly demonstrate that no such monopoly of expression exists.

To begin, there are a vast number of programming languages that have been created both before and after the time Java was created. A commonly referenced list of "significant" languages (as of 2000) included fifty different ones,[115] extracted "from among the over 1,000 high-level implemented languages ... that have been defined since work in

---

111. An analogy to computer programming languages is the Klingon "language" created as part of the Star Trek franchise. *See Klingon Language in* WIKIPEDIA (last edited May 29, 2022 1:31 (UTC)), https://en.wikipedia.org/wiki/Klingon_language. Many fans of Star Trek have spent time exploring this area and can even speak Klingon, but acknowledge as they are doing so that they are an "authorized user" of the copyright that belongs to Paramount Pictures. *See Klingon Language Inst.* (last visited June 20, 2022 5:02 PM), https://www.kli.org/.

112. *See* Thimbleby, *supra* note 3, at 937.

113. *See* 17 U.S.C. §§ 102(a) & 201(a)-(b) (2020).

114. *See* Mason v. Montgomery Data, Inc., 967 F.2d 135, 138 (5th Cir. 1992).

115. Jean E. Sammet, *Appendix VI in* ENCYCLOPEDIA OF COMPUT. SCI. 1937, 1939–43. (Anthony Ralston *et al* eds., 4th ed. 2000)

computing started."[116] Of course, programming languages did not stop developing in 2000. Since then, while many "older" languages are still used,[117] many significant new languages have been created.[118] Based on the list of what is currently used, Java may be among the most popular language in use today, but it clearly not the only programming language.[119] Consequently, with over 1,000 expressed programming languages, no one of them merges as there are clearly more ways to express one.

It should be obvious, also, that there is nothing unique about Java that should cause it to merge when other programming languages do not. From a technical perspective, Java is an object-oriented language,[120] but so are many others.[121] Java allows the creation of

---

116. *Id.* at 1937.

117. COBOL, a language created in the late 1950s which operates in a significantly different way than modern programming languages, still has a significant presence in the computer world. *See* Patrick Stanard, *Today's Business Systems Run on COBOL*, TECHCHANNEL (Mar. 10, 2021), https://techchannel.com/Enterprise/03/2021/business-systems-cobol. It is estimated that there are over 200 billion lines of COBOL code in current use, often for core business systems, with over a billion more lines added annually. *See id.*

118. *See* Brian Eastwood, *The 10 Most Popular Programming Languages to Learn in 2022* (June 18, 2020), https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/. Most of the subjects in the authors' *Programming Creativity* study chose from this list using mostly Python, but Swift was also present within the dataset.

119. *Cf. id.* (listing more programming jobs in Java than any other language).

120. *See KEN ARNOLD, JAMES GOSLING, & DAVID HOLMES, THE JAVA PROGRAMMING LANGUAGE (4th ed. 2005). See generally,* Peter Wegner, *Why Interaction is More Powerful than Algorithms*, COMM. OF THE ACM, May, 1997, at 80 (describing the advantages of decreasing focus on algorithms).

121. *See* Trung Tran, *Top 6 Object-Oriented Programming Languages*, ORIENT (Dec. 17, 2021), https://www.orientsoftware.com/blog/list-of-object-oriented-programming-languages/ (listing Java along with C#, Python, Ruby, PHP, and TypeScript).

APIs,[122] but so do many others.[123] From the computer science perspective, Java is a member—granted, a very popular member—of a pack of other similar languages. As it is imminently possible for a new author to express a language like Java (because several already have), merger does not operate.

C. Conclusion: Software is a Fully Expressive Work

It is not surprising that the *Feist* case addressed the copyrightability of a telephone book's white pages.[124] Clearly that database of the last century lived just off the edge of human expression. As the Court determined, taking all data points (the names and phone numbers) and putting them into the only logical order available (alphabetical) expressed nothing that can fairly be called "creative."[125] Directly, *Feist* removes straight-forward databases and other similar non-expressively creative works from the ambit of copyright.

A computer program is not a database, however, nor is it like one. Software is not comprised of a list of mandatorily chosen data items placed into a predefined order; instead, it is a creatively written expression designed, ultimately, to be operable on a computer system (while also explaining the details of its operation to those who read its code).[126] The fact that programs can operate on a computer does not mean that they lack

---

122. *See supra* text accompanying notes 96–99.

123. *See, e.g.,* James Briggs, *The Right Way to Build an API with Python*, TOWARDS DATA SCIENCE (Sep. 11, 2020), https://towardsdatascience.com/the-right-way-to-build-an-api-with-python-cd08ab285f8f; Ajit Mungale, *C# and API*, C#CORNER (Dec. 30, 2005), https://www.c-sharpcorner.com/article/C-Sharp-and-api/; *supra* note 3.

124. Feist Publ'ns, Inc. v. Rural Tel. Serv. Co., Inc., 499 U.S. 340, 342 (1991).

125. *See id.* at 345.

126. *See supra* section II.

in copyrightable content, however. As one court said, "Although processes themselves are not copyrightable, an author's description of that process, so long as it incorporates some originality, may be protectable."[127] As the *Programming Creativity* study established, all nontrivial computer programs do this. In other words, a computer program is a description of a process to be achieved that required creativity by its programmer for its existence. It is not, therefore, a question of how little of a program is protectable by copyright; it is a question of how little of it is an idea. This, in turn, requires a careful application of the methods used to dissect the ideas from the expression in computer software.

IV. Applying the Idea/Expression Dichotomy to Computer Programs: Using the Abstraction-Filtration-Comparison Approach Appropriately

As discussed above, all nontrivial computer programs are expressive and should obtain copyright protection as a result. Having done so, of course, does not answer all of the questions raised by computer software copyrights. Section 102(b)'s idea/expression dichotomy still requires exclusion of some aspects of all computer programs (and all other works of authorship, for that matter).[128] Fundamentally, copyright requires that the ideas underlying the expressed code remain available despite the claimed copyright.[129]

---

127. Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823, 837 (10th Cir. 1993).

128. *See, e.g.*, 17 U.S.C. § 102(b) (2020); Computer Assoc. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992); Whelan Assocs., Inc. v. Jaslow Dental Lab'y, Inc., 797 F.2d 1222 (3d Cir. 1986).

129. *See* 17 U.S.C. § 102(b) (2020).

Courts have been addressing this issue in software over the last decades.[130] Not surprisingly, most courts have relied on specialized version of the typical copyright deconstruction analysis[131]—the abstraction-filtration-comparison test[132]—to coordinate this. By breaking the software down in multiple ways, the line between idea and expression can be drawn.[133] What does not work, however, is assuming that there is a simple way to do this, or that examining one aspect of a computer program will result in an accurate placement of the line. The primary example of courts doing this in an inappropriate and destructive way is when they conflate "efficiency" with a lack of creative expression.[134] Under this approach, if "efficiency" is found within the software, the program is to be treated as an idea rather than an expression under section 102(b).[135]

---

130. *E.g.* Whelan Assocs., Inc. v. Jaslow Dental Lab'y, Inc., 797 F.2d 1222 (3d Cir. 1986); Computer Assoc. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992); Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823 (10th Cir. 1993); Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996); MiTek Holdings, Inc. v. Arce Eng'g Co., 89 F.3d 1548 (11th Cir. 1996); Dun & Bradstreet Software Servs., Inc. v. Grace Consulting, Inc., 307 F.3d 197 (3d Cir. 2002); Gen. Universal Sys., Inc. v. Lee, 379 F.3d 131 (5th Cir. 2004).

131. *See, e.g.*, Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (1930) (L. Hand, C.J.).

132. *See, e.g.,* Computer Assoc. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 706–12 (2d Cir. 1992).

133. *See id.*

134. *See* Computer Assoc. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 708 (2d Cir. 1992); Merch. Transaction Sys., Inc. v. Nelcela, Inc., No. CV02-1954-PHX-MHM, 2009 WL 723001, at *14 (D. Ariz. Mar. 18, 2009); CSS, Inc. v. Herrington, No. 2:16-CV-01762, 2017 WL 3381444, at *9 (S.D. W. Va. Aug. 4, 2017).

135. *See* authorities cited, *supra* note 134.

The reality is that this use of what the courts call "efficiency" is based on a false assumption: that there is such a thing as a singular most efficient computer program to solve a particular problem.[136] In reality, efficiency itself is an amorphous concept—indeed, a utopian goal, at best—within all engineering-based disciplines including computer science.[137] As Professor Petroski describes it, "Designing anything, from a fence to a factory, involves satisfying constraints, making choices, containing costs, and accepting compromises."[138] There is no single point of efficiency for any engineered project, including software.

Unfortunately, when examining use of efficiency within the case law examining computer software, the existence of "constraints, ... choices, ... costs, and ... compromises"[139] in creating the program are not incorporated into the decision-making.[140]

---

136. Even the legal literature has long acknowledged that computational efficiency is not the primary goal of most computer software creation. *See* Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 STAN. L. REV. 1045, 1052 (1989). What has not been recognized, however, is that efficiency, itself, is unobtainable.

137. *See* HENRY PETROSKI, SMALL THINGS CONSIDERED 4–13 (2003); David Hemmendinger, *Procedure-Oriented Languages in* ENCYCLOPEDIA OF COMPUTER SCI. 1441 (Anthony Ralston *et al* eds., 4th ed. 2000) (describing how these "higher-level" languages make coding faster and allow for less hardware dependency—both forms of efficiency—even though assembly language would operate more directly and speedily—an alternate form of efficiency). *See generally* SHERIF D. EL WAKIL, PROCESSES AND DESIGN FOR MANUFACTURING 10–12 (2d ed. 2002) (describing engineering trade-offs and stating that "the most efficient design ... is ... the one that would be favored by the customers and/or the society as a whole"); *Code Efficiency*, TECHOPEDIA (MAR. 14, 2017), https://www.techopedia.com/definition/27151/code-efficiency.

138. PETROSKI, *supra* note 137, at 13.

139. *Id.*

140. *See* Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 708 (2d Cir. 1992).

The court, often triggered by the party challenging the copyright, determines that the program is efficient so it must be an idea under section 102(b).[141] To both avoid this and understand why it can never succeed, the complexity of defining efficiency within a computer program must be appreciated.[142] In fact, efficiency within a program can be defined in numerous, but inconsistent ways, the more common of which are described next.

1. A Non-Exhaustive List of Efficiency Definitions

Speed of Execution: Most times, judicial discussions of finding efficiency in a computer program appears to be alluding to a determination that would calculate the number of computer instructions that will need to be executed to complete its task.[143] Apparently, if this number is smaller than other ways of programming the computer, this version of the program would be found to be the most efficient and would be excluded from copyright protection under the determination that it is an idea.[144]

In reality, doing this cannot work. Simply counting the number of instructions executed by a program is not meaningful as different basic instructions (such as multiplication and addition) take a different amount of time to execute.[145] Even if this were

---

141. *See id.*

142. *See* Patricia B. Van Verth, *Software Metrics in* ENCYCLOPEDIA OF COMPUT. SCI. 1627, 1628–30 (Anthony Ralston *et al* eds., 4th ed. 2000) (describing multiple ways of "measur[ing] ... software").

143. *See, e.g., Computer Assocs.*, 982 F.2d at 708.

144. *See id.*

145. *See Is multiplication slower than addition on modern CPUs?*, RESEARCHGATE (Feb. 23, 2018), https://www.researchgate.net/post/Is-multiplication-slower-than-addition-on-

not true, there is no easy way to determine how many instructions will need to be executed just by examining the algorithm. For most—except the most trivial ones—it is mathematically impractical (or, for some algorithms, impossible) to calculate the number of steps that will be needed.[146] As importantly, for many algorithms, the actual data being processed will affect the number of execution steps needed, potentially radically.[147] Thus, as a general matter, any calculation of the speed in this way will be accurate only for the particular variation of data to be processed.[148]

To avoid this problem, a proxy for speed of execution could be obtained by determining the amount of clock-time that is needed for the program to complete known

---

modern-CPUs (comparing the speed of adding and multiplying numbers and estimating that multiplication takes three times longer). *See generally* Dennis J. Frailey, *Computer Architecture in* ENCYCLOPEDIA OF COMPUT. SCI. 304, 313–15 (Anthony Ralston *et al* eds., 4th ed. 2000) (discussing the various ways computers have been designed); Shmuel Winograd, *How Fast Can Computers Add?*, SCI. AM., Oct., 1968, at 93 (detailing how the electronics underlying mathematical operations work).

146. *See* DONALD E. KNUTH, FUNDAMENTAL ALGORITHMS 7 (2d ed. 1973); *id.* at 10–92 (describing the mathematical methods needed to evaluate an algorithm); *id.* at 94–102 (analyzing the *average* execution speed of a simple method of determining the largest number in a list).

147. *See id.* at 95–96.

148. *See id.* at 96 (stating that the determination of execution speed of an algorithm will result in a minimum, maximum, and average value determined by the dataset being processed); DONALD E. KNUTH, SORTING AND SEARCHING 73–75 (1973) (stating that about twenty-five methods of putting data into a set order will be discussed, each having its own "advantages and disadvantages"). As an example, using an "insert-sort" algorithm (the technique most card players use to order their hand by suit and rank) is generally considered one of the slowest ways for a computer to place information in order, but it will excel if the data are already highly ordered. *See id.* at 110.

as "bench-marking,"[149] but this produces inconsistent results for two primary reasons. First, as before, the data used in the bench-marking attempt will affect the results, often dramatically.[150] Second, the multitasking nature of the modern computer will affect the accuracy of the results.[151] With a multitasking computer, the other processes that are active during each bench-marking attempt will produce widely variable bench-marking results.[152] Dedicating the computer to a particular task to avoid this leads to an equally unreliable result as modern computers are always multitasking,[153] so the measured benchmark will be significantly different than reality, particularly if the program being measured used a commonly needed secondary resource—the main data storage disk being the primary example— as the competition for this resource will significantly raise the bench-marked result.[154]

---

149. *See* Rudi Eigenmann, *Benchmarks in* ENCYCLOPEDIA OF COMPUT. SCI. 137 (Anthony Ralston *et al* eds., 4th ed. 2000).

150. *See id.* at 139.

151. *See id.* at 137–39.

152. *See* Walter F. Tichy, *Multitasking in* ENCYCLOPEDIA OF COMPUT. SCI. 1210 (Anthony Ralston *et al* eds., 4th ed. 2000) (describing how a multitasking computer shares its resources among various programs that appear to be running simultaneously). Almost all modern computers, from cell phones to super computers, multitask. *See id.*

153. Many modern computers have multiple processors which can avoid competition at that level unless more tasks are active than the number of processors. *Id.* at 1210. When this happens, one or more of the processors must be shared, called "multiplexing." *Id.* If multiplexing is needed, the bench-marking results will be much higher. *See Id.*

154. *See id.* This problem results particularly if a physical, spinning hard disk is used as the latencies cause by disk rotation and head movement are significant limitations on execution speed. *See* David N. Freeman, *Access Time in* ENCYCLOPEDIA OF COMPUT. SCI. 8, 8–9 (Anthony Ralston *et al* eds., 4th ed. 2000). The issue also comes up, although it is of smaller magnitude, for the newer silicon "disks" that have no rotational or head-seek

In addition to being very difficult to measure, the basic speed of execution is most often irrelevant in the real world of software design. Unless a developed program operates too slowly, the other "efficiency" considerations discussed below are of higher import.[155] As long as the program is "fast enough," there is no concern about increasing the speed more.

Technology Needed: In most ways, the decision about what technology (particularly hardware) is available on which to execute the software will be more determinative of overall program speed than anything else. If the hardware that is available is primitive, the software written for it will need to be highly limited in its functionality or it will operate at unacceptably slow speeds.[156] Equally, if the hardware is powerful, much more enabled

---

delays because the interface they use to be compatible with all software introduce significant delays as does the slower operating speed of the memory chips used. *See* Stephen J. Rogowski, *Hard Disk in* ENCYCLOPEDIA OF COMPUT. SCI. 767, 768 (Anthony Ralston *et al* eds., 4th ed. 2000).

155. In the study of computer programmer creativity report on above, for example, each programmer was required to produce code that would provide an answer within a maximum of five minutes of computer clock-time, but had no incentive to produce code that was faster than this. *Programming Creativity, supra* note 17, at 4. In the computer science-trained authors' collective experience, this is a common approach to execution speed, particularly in commercial software development. All users want their software to be faster, but none seem willing to pay the price that would be needed for the speed to be provided.

156. A simple comparison between the processing done by a standard piece of consumer electronics (a PC, cell phone, or automobile interface, as examples) and the kind of technology used by online businesses is illuminating. Most have been surprised by the predictive technology that attempts to determine what other music, movie, or other product we would like to buy or, similarly, shows an advertisement that seems to know what we have been doing recently. The computer processing necessary to do this is intensive, usually required large arrays of multiple computers so that the billions on lines of data can be processed quickly enough. *Cf. Top 18 Advertising Analytics Software*, PAT RESEARCH (last visited July 23, 2022 11:48 AM), https://www.predictiveanalyticstoday.com/top-advertising-analytics-software/. If an attempt were made to do this on consumer-level computer technology, it would take too long to finish. The automobile interface would be particularly bad at this as, it seems, most systems do not have the capacity to activate a back-up camera and turn off the music

software becomes possible.[157] Consequently, the computer system's maximum capability is more determinative of execution speed than how the programmer has chosen to write the software.[158]

Of course, the decision point for what hardware is available can be reduced to a common problem: the amount of money available for the project. Very capable hardware carries a large price tag.[159] Consequently, rather than being a way of distinguishing between an expression and its underlying idea, establishing computer program efficiency may be better appreciated as merely a measurement of the wealth of the entity producing or running it.

---

system at the same time.

157. This increase in capabilities is often accompanied by a *decrease* in the speed of execution. *See* Niklaus Wirth, *A Plea for Lean Software*, 28 COMPUTER, Feb., 1995, at 64.

158. *Cf. id.* Also, this is not a stable determination over time. Programs that are run routinely today—machine learning or weather forecasting, as examples—would be incapable of operation on the computers of the last century. *Cf.* NAT'L RSCH. COUNCIL, THE FUTURE OF COMPUTING PERFORMANCE: GAME OVER OR NEXT LEVEL at 55 (2011) (showing thousand-fold increase in computer performance between 1985 and 2010). Likewise the software developed for the first generation of widely available personal computers would be seen today as amusing examples of incompetence even though the execution speed would be quick. For example, it is hard to imagine that Wordstar; an early (and market dominating) non-WYSIWYG, micro-based word processor; would successfully process the complexities of a typical law review article, particularly if the user of the software is not sophisticated in the operation of a computer. *See* Winword, WordStar 0.x/1.x, https://winworldpc.com/product/wordstar/0x1x (last visited Apr. 18, 2022).

159. *Cf.* Paul E. Ceruzzi, *Digital Computers Since 1950 in* ENCYCLOPEDIA OF COMPUT. SCI. 552 (Anthony Ralston *et al* eds., 4th ed. 2000).

Cost of Producing the Code: Another way to measure the efficiency of computer code is to evaluate the cost of producing it.[160] If developing the program is beyond the resources of a company, no other efficiency has meaning other than looking for ways to minimize or eliminate these costs; indeed, one can posit that this was a motivation of Google in the *Google v. Oracle* case.[161] Google knew that programmers could be hired who already knew how to use Oracle's Java system with its APIs.[162] By appropriating these APIs, the efficiency of creating the Android programs were greatly enhanced, making it more "efficient" than other options by being significantly cheaper to produce.[163] This, of course, had nothing to do with how well the APIs operated and seems to be completely irrelevant to whether the APIs are expressions or ideas.[164]

Another example of this factor was the development of the "very high level languages" ("VHLL") and other nonprocedural programming techniques starting in the late 1960s and early 1970s.[165] These programs were designed to simplify programming by

---

160. *See* FREDERICK P. BROOKS, JR., THE MYTHICAL MAN-MONTH (Anniversary ed. 1995). As Professor Brooks discusses, even defining these costs are extraordinarily difficult. *See, e.g., id.* at 4.

161. *See* Google LLC v. Oracle America Inc., 141 S. Ct. 1183, 1190 (2021).

162. *See id. ("*Google wanted millions of programmers, familiar with Java, to be able easily to work with its new Android platform....").

163. *See id.*

164. The reality is, of course, that the APIs, like all other copyrighted works, are a combination of both. *See, e.g.*, Nichols v. Univ. Pict. Corp., 45 F.2d 119 (1930) (L. Hand).

165. *See* Burton M Leavenworth, Jean E. Sammet, & David Hemmendinger, *Non-procedural Languages in* ENCYCLOPEDIA OF COMPUT. SCI. 1244 (Anthony Ralston *et al* eds., 4th ed. 2000).

removing any concern for the details of the underlying machine's operation, being problem-oriented rather than procedurally-oriented.[166] Some of these VHLLs keep the basic procedural predicates that have been used since the early years of computer programming but remove the need for the programmer to understand how the underlying computer hardware works[167] while others remove the predicates, too, and just require English-like statements of the desired result.[168] But there is a dark side to these VHLLs: the operating program will be significantly slower in operation and require significantly more hardware capability than a similar program written in a harder to use language.[169] In other words, coding ease and efficiency comes at the expense of operational efficiency: if you have one you cannot have the other.

---

166. *See id.*; C. WILLIAM GEAR, COMPUTER ORGANIZATION AND PROGRAMMING 14–16 (1969).

167. A popular example here would be Java which places its procedural aspects within "methods" which are then easily reused. *See* Thimbleby, *supra* note 3, at 937.

168. *See* Leavenworth, *supra* note 165, at 1245. A prime example here is Focus, *see Focus*, WIKIPEDIA (Feb. 13, 2022 14:28 UTC), https://en.wikipedia.org/wiki/FOCUS, but the command structure found within Excel would also qualify; indeed, the automated voice response systems such as Alexa/Assistant/Cortana could also be considered the ultimate goal of this kind of software. For a general discussion of the current status of computers understanding spoken language, see Hang Li, *Language Models: Past, Present and Future*, COMM. OF THE ACM, July, 2022, at 56.

169. *See* Mark Gibbs, *A High-Level Language Worthy of Your Tool Kit*, NETWORK WORLD, June 14, 1999, at 44 ("if you want raw speed, [VHLL] aren't the way to go."). For Alexa/Assistant/Cortana, the needed computer power is somewhat hidden from the user as the real processing power, being well beyond anything located in most people's house, is handled by a cloud-based (what used to be called distributed) computing service. *See* Richard Baguley & Colin McDonald, *Appliance Science: Alexa, how does Alexa work? The science of the Amazon Echo*, CNET (Aug. 4, 2016 5:00 AM PT), https://www.cnet.com/home/smart-home/appliance-science-alexa-how-does-alexa-work-the-science-of-amazons-echo/.

Ease of Modification: Another way of defining efficiency is to determine how easy it is to modify the software for new or changing purposes.[170] Most modern software is revised on a regular basis.[171] A good example of this would be any of the tax preparation software that is on the market.[172] As tax laws change on a regular basis, making it possible to easily incorporate the new laws becomes an efficiency consideration for the company; indeed, this consideration is likely to be predominant than other efficiency factors.[173] Most times, however, this ease of modification results in a computer program that is computationally slower.[174]

Reducing Software Bugs: A recurrent problem in computer science is developing ways that error-free software can be produced.[175] Among the techniques discussed by

---

170. *See* Marvin Zelkowitz, *Perspectives of Software Engineering*, 10 COMPUTING SURVEYS 197 (1978) (discussing the life-cycle of computer software systems); Videla, *supra* note 28, at 56–58; Benjamin Mittman & Jean E. Sammet, *Problem-Oriented Languages in* ENCYCLOPEDIA OF COMPUT. SCI. 1433 (Anthony Ralston *et al* eds., 4th ed. 2000) (discussing the advantages of programming approaches that emphasize non-technical coding to enable ease of programming).

171. *See, e.g.*, *Microsoft Word*, MICROSOFT WIKI (last visited Apr. 20, 2022 10:42 AM), https://microsoft.fandom.com/wiki/Microsoft_Word (listing at least 30 versions of the Word program). *Cf. supra* note 117 (discussing the current use and modification of business software).

172. *See, e.g., Manually Update TurboTax Business Software*, TURBOTAX (last visited June 22, 2022 2:07 PM), https://ttlc.intuit.com/turbotax-support/en-us/help-article/update-products/manually-update-turbotax-business-software/L3WqYBBgc_US_en_US (describing annual updates).

173. *Cf. id.*

174. *See* Mittman & Sammet, *supra* note 170 (recognizing that non-technical coding produces programs that often take significantly longer to execute).

175. *See* BROOKS, *supra* note 160, at 142–50.

Professor Brooks is the use of top-down design and structured programming to develop new software.[176] By imposing these artificial organizations on the developing code, many mistakes in coding can be avoided in the first place or, at least, made easier to find and repair as the development continues.[177] Similarly, the recent development and use of object-oriented programming, including Java, further abstracts the coding process by defining specific fields with their associated attributes combined with methods for processing the data.[178] The ability to program at this higher level of abstraction is not free, however, as the implemented code will almost certainly need more clock-time to operate.[179]

Interoperability: An early and continuing issue in developing computer programs was the lack of interoperability. A program that was written to run on an IBM mainframe would not operate on another company's machine, e.g., Honeywell or Digital Equipment.[180]

---

176. *Id.* at 143–44.

177. *See id.* These techniques have provided some help in producing error-free code, but the problem persists, partially because computer languages, like other human-created languages, contain fundamental ambiguities. *See* Alvaro Videla, *Meaning and Context in Computer Programs*, COMM. OF THE ACM, May, 2022, at 56.

178. *See* Peter Wegner, *Object-Oriented Programming (OOP) in* ENCYCLOPEDIA OF COMPUT. SCI. 1279 (Anthony Ralston *et al* eds., 4th ed. 2000).

179. *See* BROOKS, *supra* note 160, at 143 (discussing top-down design's use of modules); Luca Cardelli, *Bad Engineering Properties of Object-Oriented Languages*, ACM COMPUT. SURV., Dec., 1996, at 28 (discussing object-oriented coding). To implement these programming approaches, multiple subroutine calls are normally used. *See* Adrienne Bloss & J.A.N. Lee, *Subprogram in* ENCYCLOPEDIA OF COMPUT. SCI. 1708 (Anthony Ralston *et al* eds., 4th ed. 2000). Each call has a small execution speed overhead which can be significant if the module is used often. *See* Edwin D. Reilly, *Calling Sequence in* ENCYCLOPEDIA OF COMPUT. SCI. 193, 194 (Anthony Ralston *et al* eds., 4th ed. 2000) (describing the process steps needed to pass a parameter to a subroutine).

180. *See* Paul E. Ceruzzi, *History of Digital Computers Since 1950 in* ENCYCLOPEDIA OF COMPUT. SCI. 552, 554–55 (Anthony Ralston *et al* eds., 4th ed. 2000).

Similarly, programs developed for the first microcomputers that were based on the 8080/Z80 family of chips and the CP/M operating system would not operate on the 8088/8086 hardware family using PC-DOS (and now Windows) that was found in IBM-branded microcomputers.[181] Of course, if programs routinely could operate on alternate hardware, the efficiency of not having to reprogram the application for each hardware platform would be gained.

Indeed, the Java language underlying the *Google* case became popular substantially because it overcomes much of the interoperability problem.[182] One powerful part of Java is know as the Java Virtual Machine.[183] Until this century, most computer languages would compile into the native machine language for a particular computer or would be interpreted by software that could only run on a particular type of hardware.[184] More recently, languages such as Java do not do this, producing instead an "intermediate

181. *See* Larry D. Wittie, *Microprocessors and Microcomputers in* ENCYCLOPEDIA OF COMPUT. SCI. 1161, 1166 (Anthony Ralston *et al* eds., 4th ed. 2000). To a certain extent, this incompatibility was artificially created as a marketing decision when the IBM PC was first released to help IBM capture a majority of the developing microcomputer marketplace but also owed its origin, as many business decisions do, to happenstance. *See* Jeremy Reimer, *Total Share: 30 Years of Personal Computer Market Share Figures*, ARSTECHNICA (Dec. 15, 2005, 12:00 AM), https://arstechnica.com/features/2005/12/total-share/.

182. *See* Thimbleby, *supra* note 3, at 937. This is not to minimize the importance of its object-oriented approach to programming that makes it easier to develop the code, particularly Internet- and Web-based programs, in the first place. *See id.* at 938–39.

183. *Id.* (describing the Java Virtual Machine).

184. *See, e.g.,* IBM SYSTEM/360 OPERATING SYSTEM PL/I (F) COMPILER: PROGRAM LOGIC MANUAL 13–15 (1966), www.bitsavers.org/pdf/ibm/360/pli/ Y28-6800-1_PL1(F)_PLM_Sep66.pdf.

language" version.[185] This allowed new parties who wished to use Java on different hardware to write their own version of the Virtual Machine—a much smaller programming task than rewriting all of Java—which could then execute any Java code on the different computer.[186] Java's efficiency, therefore—indeed, maybe its key efficiency—is its high level of interoperability. The required trade-off is also there: no one has ever accused Java of executing quickly.[187]

2. The Use of Efficiency as a Measure of Copyrightability Fails Causing Worthy Programs to be Unprotected

These examples of the different kinds of efficiency that can be found in computer software demonstrates the fallacy in asking whether a piece of software is "efficient." Even that question standing alone is nonsensical. Going farther and attempting to use it to separate computer software expression from idea compounds the problem.

A similar type of question would be to ask whether a particular bridge across a river is the most "efficient" way of building one.[188] Its designers certainly had to consider traffic (both quantity and weight) using the crossing, but also had to incorporate a wide range of

---

185. *See* Ron Cytron, *Intermediate Languages in* ENCYCLOPEDIA OF COMPUT. SCI. 910 (Anthony Ralston *et al* eds., 4th ed. 2000). Importantly, of course, other languages other than Java have used the intermediate language technique without appropriating any of the Java Virtual Machine language. *See, e.g.,* PETER TRÖGER, PYTHON (2.5) VIRTUAL MACHINE: A GUIDED TOUR, (Apr., 2008), http://www.troeger.eu/teaching/pythonvm08.pdf.

186. *See* Cytron, *supra* note 185.

187. *See When Is Java Faster Than C++?,* FORBES (May 26, 2015, 01:26 PM), https://www.forbes.com/sites/quora/2015/05/26/when-is-java-faster-than-c/?sh=20e2f17e3100 (estimating that Java executes three times slower than C++).

188. In literature, the same point can be raised by asking if Ernest Hemingway's writing is an idea while James Joyce's is not because Hemingway writes more "efficiently" by using fewer words than Joyce does.

other consideration—often including concerns not dictated by the engineering such as local and national politics—before a design could be established. Similarly, programs, like other engineering projects, can only be measured against the criteria individually established for them. As all engineering requires different characteristics, advantages, and costs to be set-off against each other,[189] there is no single way to measure "efficiency." To make measuring "efficiency" a critical component of determining the nature of computer software, therefore, is ridiculous.

Also, this again demonstrates the complexity of expression that underlies a computer program. Programmers do not seek some mystical point of efficiency; instead, like all other engineers, they seek to find the balance of considerations that produce a functioning program (as defined by the user) within the bounds of the numerous technological factors under which they operate.

The protocol of using the levels of abstraction within a computer program to find the magical copyright line between an idea and its expression is beyond reproach. It has worked for most copyrighted works and will work for programs. To expect this analysis to be factually simple and reducible to a single factor is not realistic. Computer software is like other copyrighted works that require considerable effort to separate the ideas from the protected expression. Short-changing the effort by only focusing on a false notion of efficiency will leave important parts of the software expression unprotected.

---

189. *See* PETROSKI, *supra* note 137, at 13.

V. Conclusion

Determining what aspects of a computer program are copyright protected and what ones are not will present factually hard problems. This, of course, has been true for most other copyrighted works, too. For a work of literature, for example, the actual words used in the work clearly are protected in their literal form while the bare plot upon which it is based is not.[190] Any author can decide to write a work based on the "wrong two people meet and come to tragic ends" plot but they cannot grab the latest romance novel version of this plot and engage in wholesale verbatim copying.[191] The factually difficult questions come when the second author does not copy verbatim, but does copy enough of the first author's expression so that the essence of the expression has been appropriated.[192] In the same way, the programmer who creates a system of APIs certainly should obtain protection for the literal code but cannot complain if another programmer independently creates another API system.[193] If the second programmer takes the expressive essence of the first programmer's API system and recreates that, non-literal copying has occurred and copyright liability would seem to be appropriate.[194]

---

190. PAUL GOLDSTEIN, GOLDSTEIN ON COPYRIGHT § 9.1.1 (3d ed. 2022-2 supp. 2022) (establishing infringement as being clear when "the defendant's 300-page novel track[s] the plaintiff's 300-page novel word for word").

191. *See* Mazer v. Stein, 347 U.S. 201, 217–18 (1954).

192. *See, e.g.*, Salinger v. Random House, Inc., 811 F.2d 90, 98 (2d Cir.), *opinion supplemented on denial of reh'g*, 818 F.2d 252 (2d Cir. 1987).

193. *Cf. id.* (involving non-software).

194. *See, e.g.*, Johnson Controls, Inc. v. Phoenix Control Sys., Inc., 886 F.2d 1173, 1175–76 (9th Cir. 1989).

Most contested copyright litigation is not easy. In literature-based cases, for example, the whole work is not taken verbatim; instead, more details are added (or, more likely, left in place) to the common plot by the second author so that the new work seems to be a continuation of the old one.[195] At some point in this kind of process, too much of the original expression will be taken so that infringement will be found.[196] For computer software, this same difficulty occurs. A programmer is free to express another object-oriented computer language that relies greatly on APIs, but if it is too directly appropriated from an existing expression—Oracle's Java, for example—infringement occurs.[197]

It is a mistake, however, to conclude that the second programmer should be allowed to appropriate the literal code created by the first because section 102(b) (or the merger doctrine) always requires that result. Although computer programs are expressive of procedures to accomplish particular results,[198] they are not procedures in themselves. As has now been established, the reality of programming is such that there are many ways to express each computer procedure or algorithm, so requiring the second author to do so independently is consistent with Congress's intent to provide copyright protection for

---

195. *Cf.* Warner Bros. Ent. Inc. v. RDR Books, 575 F. Supp. 2d 513, 536 (S.D.N.Y. 2008) (finding that a lexicon of "fictional facts" that had been created by the plaintiff was infringing).

196. *See* Penguin Random House LLC v. Colting, 270 F. Supp. 3d 736, 747 (S.D.N.Y. 2017) ("copy[ing] substantial aspects of the themes, characters, plots, sequencing, pace, and settings of plaintiffs' Novels" infringes); GOLDSTEIN, *supra* note 190, at § 9.1.2 ("the hardest case is the one in which the defendant's work reflects only structural similarities to the plaintiff's—similarities in plot, incident and character in literary works....").

197. *See Johnson Controls*, 886 F.2d at 1175-76. Again, this analysis does not factor in the fair use defense.

198. *See* 17 U.S.C. § 101 (2020) (defining "computer program").

computer programs.[199] This means that courts should treat programs in a way that is similar to "regular" copyrighted work. When doing that kind of analysis, the court does not start with an assumption that the work is an idea unless it can establish itself as something more; it goes the other way and only excludes something as an idea if it is established as one.[200] So too should a program be treated as an expression, with the court only eliminating aspects of it that are within the exclusion of section 102(b).

199. *See supra* section II.

200. *See* Nichols v. Univ. Pict. Corp., 45 F.2d 119 (1930) (L. Hand).