Antitrust Implications of Protecting Software Interfaces

Gary Pulsinlli*

Abstract

Google based its Android mobile phone operating system on Oracle's Java language. It rewrote all the functional Java code, but to remain compatible with Java (and thus leverage the experience of the extensive cadre of existing Java programmers), it copied verbatim a large number of Java's API "declarations" (the code defining the names of Java's methods and the arguments they take) so that its methods would have the same names. It also copied the overall organizational structure of those methods. Oracle subsequently sued Google for copyright infringement. The Federal Circuit found Oracle's declaration code and organizational structure protectable, and it rejected Google's various defenses—particularly fair use—that this type of limited copying to maintain compatibility and interoperability should be permissible. The U.S. Supreme Court subsequently granted certiorari, and the case is currently being briefed before that Court. Many parties have filed amicus briefs on behalf of Google, including the American Antitrust Institute ("AAI"). The AAI Brief argues that the Federal Circuit's holdings "fail to consider that copyright law seeks to promote innovation and consumer welfare by preserving a balance between exclusive rights and competition," and thus "the Federal Circuit's rulings, if not overturned, may slow innovation and competition in software-dependent markets." Its particular concern is that such strong protection of software and software interfaces—which are, at heart, fundamentally functional and thus not within copyright's core—will entrench large, established firms because of network effects, switching costs, and lock-in. It suggests that the Supreme Court should overrule the Federal Circuit and instead reinvigorate "copyright law's core competition safeguards in software markets—Section 102(b) of the Copyright Act, the merger doctrine, and fair use" and thereby restore competitive balance to software-driven markets. This paper concludes that the AAI and other amici are correct that copyright law should not protect computer interfaces and therefore the Supreme Court should reverse the Federal Circuit's ruling, which was based on a profound misunderstanding of software and how it is created.

* Associate Professor, University of Tennessee College of Law. I wish to thank the organizers of the Competition Law and Intellectual Property in the Age of Platforms and New Technology conference for inviting me to participate.

Keywords: Software, Competitive balance, US Supreme Court, Federal Circuit.

Introduction

The U. S. Supreme Court is reviewing the opinions of the U.S. Court of Appeals for the Federal Circuit in *Oracle America, Inc. v. Google LLC*.[1] The Federal Circuit's opinions allowed Oracle to use copyright to protect the software Application Programming Interfaces ("APIs") in its Java interface. The case has generate wide interest in the computer science community and beyond, as evidenced by the large number of amicus briefs filed in the case, including one on behalf of the American Antitrust Institute (AAI).[2] The AAI and other amici argue that such strong protection of software and software interfaces—which are, at heart, fundamentally functional and thus not within copyright's core—is contrary to the goals of antitrust law because it will entrench large, established firms through network effects, switching costs, and lock-in.[3] The AAI Brief suggests that the Supreme Court should overrule the Federal Circuit and instead reinvigorate "copyright law's core competition safeguards in software markets—Section 102(b) of the Copyright Act, the merger doctrine, and fair use," and thereby restore competitive balance to software-driven markets.[4]

The fundamental issue underlying the case is that software looks like a written work, and therefore it seems appropriate to protect it with copyright.[5] However, the fit is uneasy, because the purpose of software is ultimately

---

[1] 750 F.3d 1339 (Fed. Cir. 2014) ("*Oracle I*"); 886 F.3d 1179 (Fed. Cir. 2018), *cert. granted* 140 S. Ct. 520 (2019) ("*Oracle II*"). The first opinion addresses the availability of copyright protection for computer interfaces; the second addresses Google's defenses, particularly fair use. The Federal Circuit had jurisdiction because the case also included patent claims. *See Oracle I*, 750 F.3d at 1353. However, for non-patent issues, the Federal Circuit applies the law of the circuit in which the district court that issued the decision lies. Here, the court was reviewing a copyright decision from the Northern District of California, and thus it purported to apply Ninth Circuit law. *See id*.

[2] Brief for the American Antitrust Institute as Amicus Curiae in Support of Petitioner, Google LLC v. Oracle America, Inc., No. 18-956 (S. Ct. Jan. 13, 2020) [hereinafter "AAI Amicus Brief"].

[3] *See id.* at 5.

[4] *Id.* at 5-6. The views and analysis expressed in the AAI Amicus Brief are largely shared by other amici that I read, as will be discussed *infra*.

[5] *See id.* at 2 (noting that "Congress extended copyright protection to software in 1980"); *cf.* 17 U.S.C. § 101 (defining "computer program" as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result"); *id.* § 117 (entitled "Limitations on exclusive rights: Computer programs").

functional, rather than expressive[6] (as one judge put it, "Applying copyright law to computer programs is like assembling a jigsaw puzzle whose pieces do not quite fit."[7]). *Oracle v. Google* brings that difficulty to the fore, as the courts struggle to figure out which parts of software are protectable and which are not. This paper will address that question by examining the merits of the positions of the Federal Circuit and some of the Supreme Court amici, especially the AAI.

Technical Background

When Google set out to develop its Android mobile phone operating system, it decided to start with Oracle's Java programming language.[8] Google wanted to start with Java because it desired to tap into Java's extensive base of existing programs and (even more importantly) programmers.[9] Google reached out to Oracle to negotiate for a license but the parties were unable to come to terms.[10] Still wanting to use Java but also wanting to avoid copyright infringement, Google then rewrote all the Java code, creating its own libraries that had the functionality but not the code from the original language.[11] However, it was concerned that if it renamed Java's existing API[12] methods in its new system, the existing Java programmers might be unwilling to develop

---

[6] *See* AAI Amicus Brief, *supra* note 2, at 3.

[7] *Lotus Dev. Corp. v. Borland Int'l, Inc*., 49 F.3d 807, 820 (1st Cir. 1995) (Boudin, J., concurring). This passage is quoted both by the Federal Circuit in its first opinion, *see Oracle I*, 750 F.3d at 1354, and by the AAI in its brief, *see* AAI Amicus Brief, *supra* note 2, at 3.

[8] *See Oracle I*, 750 F.3d at 1350.

[9] *See id.* (noting that "Google believed Java application programmers would want to find the same . . . functionalities in the new Android system callable by the same names as used in Java."): *see also* Brief Amici Curiae of 83 Computer Scientists in Support of Petitioner at 3, *Google LLC v. Oracle America, Inc*., No. 18-956 (S. Ct. Jan. 13, 2020) [hereinafter "Computer Scientists Amicus Brief"] ("Programmers could use their preexisting knowledge to simultaneously write Java libraries for both desktops and smartphones.").

[10] *See Oracle I*, 750 F.3d at 1350.

[11] *See id.* at 1350-51.

[12] As noted by one of the amici in the case, this term is problematic in that it no longer has a fixed meaning in the computer science community. *See Amicus Curiae* Brief of Developers Alliance in Support of Petitioner at 4 n.7, *Google LLC v. Oracle America, Inc.,* No. 18-956 (S. Ct. Jan. 13, 2020) [hereinafter "Developers Alliance Amicus Brief"]:

The acronym "API" is probably the most misused term in the long record of this case, and we use it here with great trepidation for fear of adding to the confusion. Such is the danger of trying to translate between two dialects (the exact point of this brief). In this brief, "API" is an umbrella term without clear definitional boundary—a concept, versus a thing. An API declaration is a thing (defined herein), distinct from implementing code (a different thing, and also defined).

applications for Android, as they would be resistant to learning a new set of commands. Therefore, to maintain compatibility with Java, Google copied the "declarations" (the code defining the names of the API methods and the arguments they take[13]) from 37 of the packages found in Oracle's Java so that its methods would have the same names.[14] Furthermore, Google had to copy Oracle's declaration code verbatim for it to be useful.[15] For the same reason, Google also copied Java's extensive organizational structure for these APIs, as the programmers needed to be able to find them before they could use them.[16]

The details of what Google actually did are important to understanding the case. The Java language itself contains a relatively small number of basic commands.[17] However, to make the language more useful to developers, Oracle also built modules from those commands to perform certain more complicated tasks.[18] These modules are what the case refers to as the APIs.[19] As an example, the Federal Circuit and some of the briefs use the method max, which takes as arguments two variables and returns the value of the greater one.[20] Each individual method comprises a declaration that lays out its name and parameters, and then the Java code that actually performs the function.[21]

Oracle wrote a large number of these API method modules and then collected them in a hierarchy consisting of three tiers: packages, classes, and methods.[22] The methods are the APIs themselves (the Computer Scientists Amicus Brief

[13] *See id.* at 4 ("As used here, an API declaration is a short line of code that is part symbolic logic, part syntax, part symbolic notation, and part pseudo-English."); *see also id.* at 5 ("An 'API declaration' as defined above would be easily understood by any developer, but the term itself is not used in the industry.").

[14] *See Oracle I*, 750 F.3d at 1350-51.

[15] *See id.*

[16] *See id.* at 1351; *see also* Developers Alliance Amicus Brief, *supra* note 12, at 14 ("API libraries make it easier to search for or call a particular API and understand its form and limitations, and they help developers understand the relationship between various APIs.").

[17] *Cf.* Computer Scientists Amicus Brief, *supra* note 9, at 7 n.4 ("While the boundary between the language and the API is fuzzy, the language is broadly responsible for defining the syntax and keywords programmers use to write software").

[18] *See Oracle I*, 750 F.3d at 1348-49.

[19] *See id.*

[20] *See id.* at 1349-50; Developers Alliance Amicus Brief, *supra* note 12, at 4, 12-13; Computer Scientists Amicus Brief, *supra* note 9, at 5.

[21] *See* Computer Scientists Amicus Brief, *supra* note 9, at 5-10; *Oracle I*, 750 F.3d at 1349.

[22] *See Oracle I*, 750 F.3d at 1348-49; Computer Scientists Amicus Brief, *supra* note 9, at 10-12.

analogizes these to the lines in a text document), which are then collected in classes of related methods (files containing multiple lines), which are in turn organized into packages (folders containing multiple files).[23] Any given method can then be called by giving its location as package.class.method.[24] The parties refer to the way this hierarchy is structured as its 'Structure, Sequence, and Organization,' or 'SSO' (a term of art in the software copyright area).[25] The Federal Circuit noted that "By 2008, the Java platform had more than 6,000 methods making up more than 600 classes grouped into 166 API packages."[26]

When Google went to develop Android, it realized that it would need to develop its own set of APIs to make the language useful for mobile phones.[27] It also desired to use the Java names for many of its methods, because those were the ones with which Java programmers were familiar;[28] however, it also knew that the actual code that Oracle used to implement the methods was clearly protected by copyright.[29] Google therefore rewrote all of the actual code used by Android.[30] However, to maintain compatibility for programmers, it had to use the same declarations as Java did, so it had to copy each of those lines of declaring code verbatim.[31] For the same reason, it had to maintain the same SSO for those methods so that programmers would know where to find them.[32] In the end, it copied verbatim the declarations and the exact structure

---

[23] Computer Scientists Amicus Brief, *supra* note 9, at 11. The Federal Circuit adopted a slightly different analogy from the District Court: "Oracle's collection of API packages is like a library, each package is like a bookshelf in the library, each class is like a book on the shelf, and each method is like a how-to chapter in a book." *Oracle I*, 750 F.3d at 1349.

[24] Here is an example from the Computer Scientists Amicus Brief (illustrated with a picture): The method sort is in the class Arrays within the package util (short for 'utility') within the master java folder. Thus, the full location of the sort method is java.util.Arrays.sort. Computer Scientists Amicus Brief, *supra* note 9, at 10-11.

[25] *See id.* at 3, 10, & *passim*; *Oracle I*, 750 F.3d at 1351, 1356, & *passim*.

[26] *Oracle I*, 750 F.3d at 1349.

[27] *See id.* at 1350.

[28] *See id.*

[29] *See* Developers Alliance Amicus Brief, *supra* note 12, at 4 ("When developers write original software (or 'code'), there is a universal understanding that they hold protected rights in their work."); *id.* at 19 ("It is accepted in the developer community that the API declaration itself is separate from the implementing code").

[30] *See Oracle I*, 750 F.3d at 1351.

[31] *See id.* at 1350-51.

[32] *See id.* at 1351; Computer Scientists Amicus Brief, *supra* note 9, at 10-12 (discussing the importance of preserving the SSO).

[33] *See Oracle I*, 750 F.3d at 1353.

of 37 of Java's API packages, for a total of 7000 lines of copied code.[33]

Here is an example to help clarify the relationship between Android and Java. The method max returns the greater of two input numbers.[34] The method in Java might look like this:

```
public static int max(int x, int y) {
    if x > y then
            return x;
    else
            return y;
}
```

The declaration for the method takes the form public static int max(int x, int y).[35] All of those words are necessary for Java to know what this is and what to do with it, except the name of the method (max) and the names of the arguments it takes (x, y).[36] The rest of the code is the implementation of the method; it is what actually compares the numbers and returns the greater.[37] When Google created Android, it would have created its own version of max:

```
public static int max(int a, int b) {
    return iif((a > b) ? a : b);
}
```

For the Android method to match the Java method, Google would have had to use the exact same declaration language, including the name of the method[38] (although it could, and generally did, use different names for the arguments, which are arbitrary[39]). However, to avoid literal infringement, Google would have written different implementing code for carrying out the method, as shown here.[40]

[34] The Federal Circuit noted that in Java this method is located at java.lang.math.max. *See id.* at 1349 (also explaining the significance and necessity of public static int).

[35] *See id.* at 1349-50.

[36] *See id.*; *see also* Computer Scientists Amicus Brief, *supra* note 9, at 8-9.

[37] *See* Computer Scientists Amicus Brief, *supra* note 9, at 9-10.

[38] *See id.* at 6-9.

[39] *See id.* at 9 ("In creating Android, Google took advantage of this creative freedom and frequently used names for inputs that differed from those in [the] Java API.").

[40] I wrote these methods myself; as far as I know, they both would work to accomplish exactly the same task, but I do not know if either matches what Oracle or Google did. The implementing code for this simple function is very short; more complicated functions require much longer implementing code.

Oracle v. Google

Oracle eventually sued Google for copyright infringement for literal copying of the 7000 lines of declaration code and non-literal infringement of the SSO of the 37 packages.[41] Google prevailed at trial, and Oracle appealed the case to the U.S. Court of Appeals for the Federal Circuit, which reversed. [42]

The Federal Circuit found Oracle's declaration code protectable because it concluded that Oracle's programmers had at their disposal a variety of ways to define these methods and place them in a hierarchy, and therefore they displayed creativity in choosing how to do so.[43] This exercise of creativity made the code protectable under copyright law. In a later decision, the court rejected Google's various arguments—particularly fair use—that this type of limited copying to maintain compatibility and interoperability for programmers should be permissible.[44] Specifically, the court held that fair use could not apply in this context unless the innovation changed the meaning or expression of the copied elements, which Google's Android had failed to do.[45]

The U.S. Supreme Court subsequently granted certiorari in the case, and it is currently being briefed before that Court.[46] Many parties have filed amicus briefs on behalf of Google,[47] including the American Antitrust Institute ("AAI").[48] The AAI Brief argues that the Federal Circuit's holdings "fail to consider that copyright law seeks to promote innovation and consumer welfare by preserving a balance between exclusive rights and competition," and thus "the Federal Circuit's rulings, if not overturned, may slow innovation

---

[41] *See Oracle I*, 750 F.3d at 1356.

[42] *See id.* at 1348.

[43] *See id.* at 1361 (declarations); *id.* at 1367-68 (SSO).

[44] *Oracle America, Inc. v. Google LLC*, 886 F.3d 1179 (Fed. Cir. 2018), *cert. granted* 140 S. Ct. 520 (2019) ("*Oracle II*").

[45] *See id.* at 1199.

[46] *Google LLC v. Oracle America, Inc.,* 140 S. Ct. 520 (2019). "The questions presented are: 1. Whether copyright protection extends to a software interface. 2. Whether, as the jury found, petitioner's use of a software interface in the context of creating a new computer program constitutes fair use." Google LLC v. Oracle America, Inc., QPReport, No. 18-956, https://www.supremecourt.gov/qp/18-00956qp.pdf.

[47] By my count, the Court has received at least 60 amicus briefs in the case. *See* Docket, Google LLC v. Oracle America, Inc., No. 18-956 (S. Ct. Oct. 19, 2018), https://www.supremecourt.gov/docket/DocketFiles/html/Public/18-956.html.

[48] AAI Amicus Brief, *supra* note 2. "The American Antitrust Institute ('AAI') is an independent non-profit organization devoted to promoting competition that protects consumers, businesses, and society. See http://www.antitrustinstitute.org." *Id.* at 1.

and competition in software-dependent markets."[49] Its particular concern is that such strong protection of software—which is, at its heart, fundamentally functional and thus not within copyright's core—will entrench large, established firms because of network effects, switching costs, and lock-in.[50] It argues that "[t]he Federal Circuit erroneously relegated several of copyright law's core competition safeguards in software markets—Section 102(b) of the Copyright Act, the merger doctrine, and fair use—to insignificance," and thereby harmed competitive balance in software-driven markets.[51] It suggests that the Supreme Court should therefore overrule the Federal Circuit and reinvigorate those limiting doctrines and thus restore the competitive balance.

The briefs of the AAI and other amici rely heavily on a principle that lies at the heart of the copyright analysis, the 'idea/expression dichotomy.'[52] A central tenet of copyright is that an author's particular *expressions* of ideas are protectable, but the underlying *ideas* themselves are not.[53] This principle is fundamental to copyright law. This dichotomy is set forth in U.S. law under 17 U.S.C. § 102(a) & (b).[54] Section 102(b) is particularly important in software cases. It provides:

> In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.[55]

This provision has at least two main purposes.[56] First, it distinguishes what belongs in the public domain from what is eligible for copyright protection.[57] Second, it establishes the boundary line between patent and copyright protection by distinguishing function and expression.[58] "The phrase 'idea/

---

[49] *Id.* at 5.

[50] *See id.*

[51] *Id.* at 5-6.

[52] *See id.* at 6, 13-14, 29.

[53] *See id.* at 13; Julie E. Cohen, Lydia Pallas Loren, Ruth L. Okediji & Maureen A. O'Rourke, Copyright in a Global Information Economy 90 (4th ed. 2015).

[54] 17 U.S.C. § 102(a) & (b). The latter in particular is a focus of the AAI Amicus Brief.

[55] *Id.* § 102(b).

[56] *See* Cohen, et. al., *supra* note 53, at 90.

[57] *Id.*

[58] *Id.*

expression distinction' is thus shorthand for a rule requiring the exclusion of a variety of elements of a work from copyright protection."[59]

Section 102(b) is important in software cases because software is often argued to be (as it was in this case) a "method of operation."[60] The AAI Brief goes so far as to assert that "the Federal Circuit failed to adequately grasp that Section 102(b) employs the idea/expression dichotomy to preserve competition and innovation by ensuring public access to the functional concepts embodied in copyrighted works."[61]

Analysis[62]

The main problem with the Federal Circuit's analysis is that it was overly impressed with Oracle's claims that it exercised a high degree of creativity in creating its declarations and SSO. I assert that the court was wrong in that analysis, for reasons that relate to two doctrines that are offshoots of the idea/expression dichotomy, merger and *scènes à faire*.

Merger

If the uncopyrightable idea underlying the protectable expression is narrow, so that the topic is necessarily limited to one or very small number of forms of expression, then copyrighting these few forms would permit coverage of idea.[63] Such an outcome is not permissible under copyright law because it would violate the idea/expression dichotomy.[64] Therefore, where only one or a few ways to express an idea are available, copyright law denies all protection of the limited set of available expressions.[65] In such a case, the expression and underlying idea are said to 'merge,' and copyright protection

---

[59] *Id.* The authors use "distinction" in preference to "dichotomy" because they believe the latter conveys a false sense of precision.

[60] *See Oracle America, Inc. v. Google LLC*, 750 F.3d 1339, 1364-68 & *passsim* (Fed. Cir. 2014) ("*Oracle I*") (discussing whether the Java SSO is a method of operation); AAI Amicus Brief, *supra* note 2, at 13-15.

[61] AAI Amicus Brief, *supra* note 2, at 13-14.

[62] I want to note up front that I am a molecular biologist by training and not a computer scientist, although I have done a fair bit of computer programming and I also teach copyright law, which gives me a decent understanding of the issues.

[63] *See* Cohen, et. al., *supra* note 53, at 96-97.

[64] *See id.*; *see also Oracle I*, 750 F.3d at 1359 (mischaracterizing merger as "an exception to the idea/expression dichotomy," when it is in fact an embodiment of that doctrine).

[65] Some courts treat merger as a bar to copyright, others as a defense to infringement. *See* Cohen, et. al., *supra* note 53, at 96-97; *see also Oracle I*, 750 F.3d at 1359 (noting that "the Ninth Circuit treats [merger] as an affirmative defense to infringement").

is inappropriate.[66] Copyright law thus does not allow anyone to cover all effective expressions of an idea.

A good example of case applying the rule is *Morrissey v. Procter & Gamble*.[67] Morrissey had created a relatively simple set of rules for running a mail-in contest using social security numbers, and Procter & Gamble later ran such a contest using a near-identical Rule 1.[68] The First Circuit found copyright protection for the rule inappropriate:

> When the uncopyrightable subject matter is very narrow, so that "the topic necessarily requires," if not only one form of expression, at best only a limited number, to permit copyrighting would mean that a party or parties, by copyrighting a mere handful of forms, could exhaust all possibilities of future use of the substance. . . . [Thus], it is necessary to say that the subject matter would be appropriated by permitting the copyrighting of its expression. We cannot recognize copyright as a game of chess in which the public can be checkmated.
>
> Upon examination the matters embraced in Rule 1 are so straightforward and simple that we find this limiting principle to be applicable. Furthermore, its operation need not await an attempt to copyright all possible forms. It cannot be only the last form of expression which is to be condemned, as completing defendant's exclusion from the substance. Rather, in these circumstances, we hold that copyright does not extend to the subject matter at all, and plaintiff cannot complain even if his particular expression was deliberately adopted.[69]

Google argued that the district court correctly held that this doctrine should apply to the declarations, and therefore they should not be protectable.[70] The Federal Circuit rejected this argument, because it believed that merger should be analyzed at the time that Oracle set out to develop Java, and at that time it

---

[66] *See Oracle I*, 750 F.3d at 1359.

[67] 379 F.2d 675 (1st Cir. 1967).

[68] *Id.* at 676-77.

[69] *Id.* at 678-79 (internal citations omitted). The Federal Circuit erroneously limited the application of the doctrine to cases where "the idea contained therein can be expressed in only one way." *Oracle I*, 750 F.3d at 1360. As explained in *Morrissey*, such a limited application of merger is inconsistent with the purpose behind the merger doctrine and the underlying idea-expression dichotomy.

[70] *Oracle I*, 750 F.3d at 1360-61.

had at its disposal a wide variety of options for naming its methods.[71] Under its analysis, merger was thus not an issue.[72] However, that analysis is deeply flawed.[73] In reality, developers are subject to significant restraints in naming and organizing a language's methods, primarily because of the practical concerns related to the conventions and expectations of computer scientists.[74]

First, the name must give *some* indication of what the method does (as one amicus noted, slight variations on max (MAX, maxi, etc.) could be "acceptable additions to a suite of APIs supporting statistical functions, since they imply a function and meaning").[75] At the same time, however, "'Aardvark' as a mathematical function might be whimsical, but it would not be embraced since it is not anchored to any deeper concept and would result in code that was non-intuitive and hard to understand."[76] Thus, in reality, only a limited number of method names would work and be accepted as function names.[77] Furthermore, even where the name is clearly linked to its function, computer programming involves a lot of typing, so programmers place a large premium on brevity.[78] They would thus infinitely prefer a method named max to one called 'whichnumberisbigger' (or even just 'bigger').

Second, the nature of computer programming is such that it would be very simple for a creator to create API declarations corresponding to a large number of variants and attach them all to the same executable code (writing

---

[71] *Id.* at 1361.

[72] *Id.*

[73] I describe some of those flaws below. For further analysis, see AAI Amicus Brief, *supra* note 2, at 16-17; Developers Alliance Amicus Brief, *supra* note 12, at 25-276.

[74] "Introducing new terms into the developer lexicon is systematized and constrained by the functional bias inherent in the nature of software development." Developers Alliance Amicus Brief, *supra* note 12, at 12; *see also id.* at 8 ("API declarations are functional by their nature, and the conventions of the software industry that created them strictly limit how they can be structured.").

[75] *Id.* at 12-13.

[76] *Id.* at 13 (further noting, "The success of any platform or language depends on how easily developers can read, remember, and wield it, and ultimately how effective it is in accomplishing programming goals.").

[77] *See id.* at 26 ("Even an original author that builds a novel programming language and a suite of API declarations is constrained in the creativity the author can wield, given the established industry norms around programming terms generally.").

[78] *See* Computer Scientists Amicus Brief, *supra* note 9, at 8 (noting that "Particularly short and intuitive names for common operations like sort become customary terms of art used across interfaces."); Developers Alliance Amicus Brief, *supra* note 12, at 23 (noting "the [computer] industry's bias towards simple and logically consistent terms and structures).

that executable code is the hard part; once that is done, duplicating it with new declaration headers is trivial).[79] The creator could thereby obtain coverage for all those variants, leaving only undesirable options for later developers.[80] This is the precise situation that the merger doctrine is intended to prevent. Thus, even accepting the Federal Circuit's premise that merger should be assessed at the time of Java's creation,[81] the practical restraints facing its programmers indicate that merger should still apply.

Scènes à Faire

The '*scènes à faire*' doctrine excludes from copyright protection "incidents, characters or settings which are as a practical matter indispensable, or at least standard, in the treatment of a given topic."[82] Such elements are thus not strictly 'necessary' to the expression of the idea (and thus do not quite merge with it), but they are so common as to be expected by the audience. And as with merger, *scènes à faire* are not protectable, because copyright law policy is strongly against allowing a single creator to tie up expression that is so tightly linked to the underlying idea.[83] Thus, much of the same analysis applied under merger also applies to *scènes à faire*: Programmers approaching a new language *expect* to find method names like max.[84] And taking the analysis a step further, Java programmers definitely expect to find a

---

[79] *See* Developers Alliance Amicus Brief, *supra* note 12, at 25 ("An enterprising developer could conceivably publish API declarations using every reasonable term that implies the 'maximum value' statistical function and point them to a single version of implementing code, effectively cornering the market for this API.").

[80] *See id.* (further noting, "This may sound extreme, but such behavior is common elsewhere, as can be seen in the market for domain names.").

[81] And as noted by the amici, that premise is in fact rather problematic. *See, e.g.*, AAI Amicus Brief, *supra* note 2, at 16 ("No product is a standard, whether *de jure* or *de facto*, at inception. When copyrighted software interfaces become standards, the efficiency derived from the interface is necessarily an emergent property.").

[82] *Hoehling v. Universal City Studios, Inc.*, 618 F.2d 972, 979 (2d Cir. 1980); *see also Oracle I*, 750 F.3d at 1363 ("The scenes a faire doctrine, which is related to the merger doctrine, operates to bar certain otherwise creative expression from copyright protection. It provides that 'expressive elements of a work of authorship are not entitled to protection against infringement if they are standard, stock, or common to a topic, or if they necessarily follow from a common theme or setting.'" (quoting Mitel, Inc. v. Iqtel, Inc., 124 F.3d 1366, 1374 (10th Cir.1997)).

[83] *Hoeling*, 618 F.2d at 979 (noting that "*scenes a faire* are not copyrightable as a matter of law").

[84] *Cf.* Computer Scientists Amicus Brief, *supra* note 9, at 8 & n.6 (noting that "Particularly short and intuitive names for common operations like sort become customary terms of art used across interfaces," and further "eight of the top ten most used programming languages . . . include a command called sort to arrange a list in ascending order").

method called max.[85] If Google wants to provide functionality that maintains compatibility for Java programmers, it will need to use the same method names, and thus in this context they are *scènes à faire*.[86]

The analysis of both merger and *scènes à faire* applies similarly to the SSO. Programmers not only expect to find certain functions in a language, they expect to find them in specific places. As noted by one of the amici, "An experienced developer can intuit the likely form of unknown but related APIs, and the overarching structure of the library used to hold them, by examining a small number of API declarations in context."[87] Thus, the organization of the SSO is based primarily on logic and efficiency, for the benefit of programmers.[88] Professor Dennis Karjala has expressed a similar sentiment:

> [A]ny court that applies the *Computer Associates* factors [in the abstraction-filtration-comparison test for analyzing software infringement] honestly will soon realize that *everything* in the SSO is present for the purpose of making the program function better, that is, for efficiency reasons. Consequently, under *Computer Associates*, after filtering for efficiency there is very little, if anything, to protect besides the code.[89]

According to Professor Karjala, after an appropriate infringement analysis that "filters out" functional elements, merger, and *scènes à faire*, all that remains of software infringement is literal infringement of the code itself.[90]

Thus, the placement of the methods into classes and then packages is not, as the Federal Circuit apparently believed, an unrestrictedly creative exercise.[91]

---

[85] *Cf. id.* at 12 ("Thus, although interface designers have some choice in naming their method declarations and inputs, programmers who reimplement an existing interface, like Google did with the Java API, *must* use the same standard names and structure to achieve interoperability.").

[86] The Federal Circuit nevertheless dismissed Google's *scènes à faire* argument, for much the same reason it rejected Google's merger argument. *See Oracle I*, 750 F.3d at 1364.

[87] Developers Alliance Amicus Brief, *supra* note 12, at 5. The Brief later notes that

The structure and characters that make up the declaration aid a knowledgeable developer in intuiting the purpose of the target API and in identifying related APIs more easily. A skilled developer, knowing what mathematical or logical function is needed for a program to perform, should quickly and easily be able to find a suitable API in its associated library. *Id.* at 14.

[88] *See id.*

[89] Dennis S. Karjala, *The Relative Roles of Patent and Copyright in the Protection of Computer Programs*, 17 J. Marshall J. Computer & Info. L. 41, 54 (1998) (citing Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992)).

[90] *See id.*

[91] *See Oracle I*, 750 F.3d at 1367-68.

The problem, I think, lies primarily in the Federal Circuit's misunderstanding of what the terms 'creative' and 'creativity' mean to a computer scientist. The court seems to assume that when a computer scientist refers to 'creativity,' he or she is using the term in the same sense that a copyright lawyer would. However, to a computer scientist, 'creativity' resides in figuring out the best and most efficient way to accomplish a task—a strongly functional undertaking rather than an artistic one (as suggested in the quote from Professor Karjala).[92] Thus, for example, the court seemed impressed with the claim by two of Java's developers that they had to make a lot of 'creative' choices in structuring the Java SSO.[93] While Oracle's programmers surely did put a lot of work into organizing the Java SSO, it was not for the purpose of making Java more 'creative' in any copyright sense.[94] Rather, the 'creative' choices they made were all in service of determining the most logical and efficient way to put things together, so that Java programmers would be able to figure out where things were with the least effort.[95] Similarly, the court noted that "Google's own 'Java guru' conceded that there can be 'creativity and artistry even in a single method declaration.'"[96] Again, however, the 'guru' was almost certainly using the terms in the computer science sense—to a computer scientist, finding a particularly effective way to construct a declaration can display "creativity and artistry" that bears little relationship to the way those terms are used in copyright law. In the final analysis, contrary to what the Federal Circuit believed, the SSO was in fact subject to rigorous restraints based on logic and the expectations of computer programmers.[97]

Fair Use

The other path argued by amici (and of course Google) is fair use.[98] The fair

[92]In the context of computer program design, the concept of efficiency is akin to deriving the most concise logical proof or formulating the most succinct mathematical computation. Thus, the more efficient a set of modules are, the more closely they approximate the idea or process embodied in that particular aspect of the program's structure.
Altai, 982 F.2d at 708.

[93] *See id.* at 1361 n.6.

[94] *Cf.* AAI Amicus Brief, *supra* note 2, at 14 (the Federal Circuit's observation that the declaring code could have been written and organized differently "is no answer to the question of whether the declaring code is unprotectable under Section 102(b) as a functional system or method of operation").

[95] *See* discussion *supra* notes 84-90and accompanying text.

[96] *Oracle I*, 750 F.3d at 1363.

[97] *See* Computer Scientists Amicus Brief, *supra* note 9, at 8-9 ("Thus, while interface designers have some choice when naming methods, the method's function, name length, and clarity constrain their choice."); Developers Alliance Amicus Brief, *supra* note 12, at 12

use doctrine in U.S. is premised on the idea that some uses of copyrighted works that might otherwise be infringements should be permitted because they serve other publicly desirable goals.[99] Unfortunately, its contours defy any kind of coherent definition, and it remains quite murky and unpredictable in its application.[100]

Congress codified fair use in the Copyright Act of 1976 at 17 U.S.C. § 107,[101] which provides a general list of types of uses generally deemed 'fair':

> [T]he fair use of a copyrighted work, including such use by reproduction in copies or phonorecords or by any other means specified by that section, for purposes such as criticism, comment, news reporting, teaching scholarship, or research, is not an infringement of copyright.[102]

The statute goes on to provide an illustrative, rather than exhaustive, list of factors to be considered in determining whether a particular use is 'fair.'[103]

However, this codification predated the wide availability of software, and

---

("Introducing new terms into the developer lexicon is systematized and constrained by the functional bias inherent in the nature of software development. A new programming language that seeks to provide full programming flexibility must remain faithful to the bias towards utility that computer science embodies.").

[98] *See* Oracle America, Inc. v. Google LLC, 886 F.3d 1179 (Fed. Cir. 2018), *cert. granted* 140 S. Ct. 520 (2019) ("*Oracle II*") (decision on fair use); AAI Amicus Brief, *supra* note 2, at 17-22 (analyzing fair use); Developers Alliance Amicus Brief, *supra* note 12, at 27-31 (same).

[99] The Supreme Court has observed:

Fair use was traditionally defined as "a privilege in others than the owner of the copyright to use the copyrighted material in a reasonable manner without his consent." . . . "[T]he author's consent to a reasonable use of his copyrighted works ha[d] always been implied by the courts as a necessary incident of the constitutional policy of promoting the progress of science and the useful arts, since a prohibition of such use would inhibit subsequent writers from attempting to improve upon prior works and thus . . . frustrate the very ends sought to be attained."

Harper & Row, Publishers, Inc. v. Nation Enterprises, 471 U.S. 539, 549 (1985) (quoting H. Ball, Law of Copyright and Literary Property 260 (1944)). The Court later stated that "The fair use doctrine thus 'permits [and requires] courts to avoid rigid application of the copyright statute when, on occasion, it would stifle the very creativity which that law is designed to foster.'" Campbell v. Acuff-Rose Music, Inc., 510 U.S. 569, 577 (1994) (quoting Stewart v. Abend, 495 U.S. 207, 236 (1990)).

[100] In *Dellar v. Samuel Goldwyn, Inc.*, 104 F.2d 661, 662 (2d Cir. 1939), Learned Hand characterized the fair use doctrine as "the most troublesome in the whole law of copyright."

[101] *See* Cohen, et. al., *supra* note 53, at 564 (noting the codification).

[102] 17 U.S.C. § 107 (2020).

[103] *Id.*

its terms have (unsurprisingly) been particularly difficult to apply in that context.[104] My own view is that the fair use case is more difficult than the idea-expression/merger/*scènes à faire* case, so I am not going to focus on its details.[105] I do, however, want to make a few points.

The first statutory factor for assessing fair use is "the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes."[106] As the analysis of this first factor has developed, the most important issue has become whether the new use merely 'supersedes' the original, or if it instead is 'transformative'—that is, whether it adds new expression and meaning to the original.[107] Furthermore, the more transformative the new work is, the more it overcomes the other factors.[108] In *Oracle v. Google*, the Federal Circuit concluded that Google was simply using the declarations in the same way that Oracle used them, and thus its use did not qualify as 'transformative' because it did not add any new expression.[109] On this point, the AAI Brief bluntly disagrees, stating that the Federal Circuit's fair use opinion "guts the fair-use doctrine as applied to software."[110] It then goes on to suggest that the Court should expand that conception and "clarify that a 'transformative use is one that communicates something new and different from the original *or expands its utility*'"[111]—an analysis other courts have applied in the software context. According to the AAI Brief, "Failing to

---

[104] *See, e.g.*, *Sega Enters. v. Accolade, Inc.*, 977 F.2d 1510 (9th Cir. 1992); *Sony Computer Entm't, Inc. v. Connectix Corp.*, 203 F.3d 596, 608 (9th Cir. 2000).

[105] The amici supporting Google do not necessarily agree with me, and I commend their briefs to the reader interested in a deeper treatment of the issue. *See, e.g.*, AAI Amicus Brief, *supra* note 2, at 17-22 (analyzing fair use); Developers Alliance Amicus Brief, *supra* note 12, at 27-31 (same). *But see* Computer Scientists Amicus Brief, *supra* note 9, at 25 (noting difficulties created if software developers are forced to rely on fair use, rather than other copyright doctrines that deny protection up front).

[106] 17 U.S.C. § 107.

[107] *See Campbell v. Acuff-Rose Music, Inc.*, 510 U.S. 569 (1994); COHEN, ET. AL., *supra* note 53, at 573-600 (discussing *Campbell* and subsequent cases applying the 'transformative' analysis).

[108] *See Campbell*, 510 U.S. at 579 ("[T]he more transformative the new work, the less will be the significance of other factors, like commercialism, that may weigh against a finding of fair use.").

[109] *Oracle II*, 886 F.3d at 1198-1202.

[110] AAI Amicus Brief, *supra* note 2, at 6.

[111] *Id.* (quoting *Authors Guild v. Google, Inc.*, 804 F.3d 202, 214 (2d Cir. 2015) (Leval, J.) (emphasis added)).

recognize utilitarian transformations in the software context is perverse, because software's benefit is primarily functional."[112] It later goes on to note that "software interfaces become standards (and are copied) because of their functional, not expressive, value."[113]

Historically, another important facet of fair use is that it is permitted for "reasonable and customary uses."[114] According to the amici I read, reuse of interface code like the API declarations in question is exactly that.[115] Indeed, the Computer Scientists Brief notes that improving the underlying functional code for existing methods is one of the primary areas of competition in the software field.[116] For example, going back to the sort method, the most obvious approach is the simplistic one of going through each item in the list to figure out which is the smallest, moving it to the top, then finding the next smallest, etc.[117] But that method is extremely slow, suitable for only very short lists, and programmers have therefore developed a large number of better, faster

---

[112] *Id.*

[113] *Id.*

[114] *See* Cohen, et. al., *supra* note 53, at 572 (discussing this "customary use" theory); *see also id.* at 622-31 (discussing this analysis as applied to computer software, under the heading "Other Production Uses"); *Harper & Row*, 471 U.S. at 550 (noting that historically, "the fair use doctrine was predicated on the author's implied consent to 'reasonable and customary' use when he released his work for public consumption").

[115] *See, e.g.*, Computer Scientists Amicus Brief, *supra* note 9, at 3-4 ("Reimplementing software interfaces is a long-standing, ubiquitous practice that has been essential to realizing fundamental advances in computing."); *id.* at 17 ("For decades, programmers have relied upon reimplementing interfaces to create fundamentally transformative technologies. Reimplementing software interfaces also promotes innovation by countering network effects and lock-in effects that inhibit competition."); *id.* at 18 ("Google's reimplementation of the Java API fits squarely within this tradition of innovation and competition."); *id.* at 19 ("It is also accepted in the industry that developers are free to innovate on both sides of the API declaration, creating new, complex programs that call existing APIs, *but also writing improved implementing code in competition with an original author*." (emphasis added)).

[116] *See id.* at 17-22 (citing as examples the reimplementation of the IBM BIOS code that led to the "personal computer revolution," the reimplementation and expansion of the original AT&T Unix operating system (and its eventual evolution into Linux), the reimplementation of the C programming language that underlies Java itself, and the reimplementations underlying the Internet and cloud computing); *see also* Brief of Amici Curiae the Computer & Communications Industry Association and Internet Association in Support of Petitioner, *Google LLC v. Oracle America, Inc.*, No. 18-956 (S. Ct. Jan. 13, 2020) (explaining how the computer industry developed largely through programmers writing interoperable code by taking advantage of unprotected interface code, such as the Java declaration code at issue in this case, and how the Federal Circuit's holding puts the progress of the industry at risk).

[117] *See* Computer Scientists Amicus Brief, *supra* note 9, at 10 (describing the "selection sort").

methods for reimplementing sort.[118]

However, if such a reimplementation is to be really effective and get readily adopted, it must be allowed to utilize the same API declarations.[119] This approach allows these types of improvements to be easily incorporated into existing programs.[120] If the new and improved implementation of a method uses the same API declaration, performance of existing software that relies on that declaration can be immediately improved, without any need for rewriting the code itself—the new method simply replaces the old in the SSO, and then the software will thereafter call the new code instead of the old.[121] If instead the declaration is protected by copyright and thus off-limits, then the author of the improved method will have to create a new declaration and thus a new name for the method.[122] It will then have to convince software developers to go back and rewrite all their existing code to call the new method in the place of the old one, which is likely to meet with strong resistance.[123] Incumbents will thereby be able to lock developers into using their proprietary code, and the pace of improvement in software will be greatly impeded.[124] Thus, this traditional model of competition in the software industry thus works only if the API declarations are free for copying—and if Oracle prevails in this case, that will no longer be true. Surprisingly, the Federal Circuit's fair use analysis

---

[118] *See id.* (noting that "Computer scientists have evaluated dozens of implementations for sort" and developed "[m]ore sophisticated implementations for sort, like 'mergesort'"); *see also id.* at 6 ("The same declaration can be implemented in various ways to accomplish the same task. Some implementations prioritize speed, others memory use.").

[119] *See* Developers Alliance Amicus Brief, *supra* note 12, at 18-20.

[120] *See id.* at 22 ("Authors of application software that relies on an API can write code confident that, should a future author come up with a better implementation of the API in question, their original software can benefit without any modification.").

[121] *See id.* ("For creators of APIs, the ability to innovate by creating a better implementation of a popular API can have an immediate impact by improving the performance of all the applications that call it.").

[122] *See id.* ("If API declarations were controlled by the original author of the implementing code, competing API authors would need to create unique API declarations of their own—even for APIs whose function was identical.").

[123] *See id.* ("In turn, application developers would need to re-write their code to swap out the old API declaration with the new one"); *see also id.* at 22-23 ("[E]ach competing API would need to find a unique API declaration for the common function from a limited list of choices that would satisfy the industry's bias towards simple and logically consistent terms and structures.").

[124] *See id.* at 23 ("Ultimately this [protecting APIs] would deter competition in implementing code and increase the effort for application developers to write code that was interoperable with many hardware environments.").

did not cite the 'reasonable and customary uses' principle, nor did any of the amici I read, but I think it is significant and relevant in this context.

Antitrust

The discussion of decreased competition in the software industry leads me into antitrust. The severe negative impact that the Federal Circuit's decision will have on competition in the software industry was a central concern of all of the amici I read. For example, in a section entitled "Allowing Copyright to Restrict the Reimplementation of Software Interfaces Will Stifle Competition,"[125] the Computer Scientists Brief observe that "The decisions below jeopardize the market for software. Reimplementing software interfaces enables startups to counter network effects and compete with established players."[126] It also notes that "Uncopyrightable software interfaces address network effect barriers by enabling startups to plug into existing systems and innovate through cumulative improvements"[127] and "extending copyright to software interfaces would enable companies to monopolize standard interfaces."[128] Similarly, in a section entitled "Software Interfaces Promote Competition and Increase Innovation by Preventing Lock-In Between Otherwise Independent Layers in Hardware/Software Systems,"[129] the Developers Alliance Brief argues that:

> If access to API declarations remains separate and independent of rights to the implementing code they represent, . . . this arrangement . . . encourages market competition by creating a mechanism for application developers to easily call on the comparable APIs of many competing implementing code authors. . . . If, on the other hand, the use and implementation of API declarations are subject to individual control, then these interfaces become a choke point for monopoly control.[130]

The AAI Brief (unsurprisingly) goes further, making antitrust its focal point. It states its position directly in its Summary of Argument:

---

[125] Computer Scientists Amicus Brief, *supra* note 9, at 22-25.

[126] *Id.* at 22.

[127] *Id.* at 23.

[128] *Id.* The brief also emphasizes that, if interfaces are copyrightable, then companies can make their interfaces free for a time until they become standards, then start imposing licensing fees. *See id.* at 23-24 (also noting that this will raise consumer prices and restrict employee mobility, and therefore "Innovation could stagnate").

[129] Developers Alliance Amicus Brief, *supra* note 12, at 18-20.

[130] *Id.* at 18. The brief further argues that "If API declarations are controlled by the author of the original implementing code, innovation would be seriously reduced." *Id.* at 20.

> Determining the copyrightability and fair use of software interfaces should be understood as an exercise in calibrating the contestability of software-driven markets controlled by dominant incumbent firms. Software pervades U.S. commerce, and when software-driven markets tip toward a single firm, affording copyright protection to software interfaces necessary for interoperability threatens to entrench such firms because of network effects, switching costs, and lock-in.[131]

The AAI Brief correctly observes that this case bears some striking parallels to an early software copyright case, *Lotus Development Corp. v. Borland International, Inc.*[132] The First Circuit's analysis in *Lotus v. Borland* serves as a very useful map for helping navigate the complex interactions among software development, copyright law, and competition, in particular recognizing the potential negative effects on competition that arise if copyright law protects software too strongly. Unfortunately, the Federal Circuit declined to follow that map, failing to recognize the anticompetitive effects its decision will create.

In *Lotus*, Lotus claimed that Borland infringed its copyright in the Lotus 1-2-3 spreadsheet program when Borland copied the Lotus 1-2-3 menu hierarchy in its competing Quattro spreadsheet program.[133] Lotus 1-2-3 was the first really popular PC spreadsheet program and thus developed a dominant position in the spreadsheet market.[134] Borland developed Quattro to overtake Lotus 1-2-3, but it knew it faced a significant problem in getting people to adopt it: Existing users were already invested in Lotus 1-2-3, and knew its interface well, and thus had no desire to change even to a vastly superior product if it meant having to learn a new interface.[135] This lock-in problem was exacerbated by the widespread utilization of user-written macros that relied on Lotus 1-2-3's detailed menu system.[136] Requiring users to rewrite all their Lotus 1-2-3 macros for Quattro (or any other new spreadsheet program)

---

[131] AAI Amicus Brief, *supra* note 2, at 5.

[132] 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 516 U.S. 233 (1996); AAI Amicus Brief, *supra* note 2, *passim* (discussing *Lotus v. Borland*).

[133] *See Lotus*, 49 F.3d at 809.

[134] *See id.*; *see also* Lotus 1-2-3, Wikipedia.org (last visited June 15, 2020), https://en.wikipedia.org/wiki/Lotus_1-2-3 (noting that "1-2-3 was the spreadsheet standard throughout the 1980s and into the 1990s," and that "[b]y early 1984 the software was a killer app for the IBM PC and compatibles").

[135] *See Lotus*, 49 F.3d at 810; *id.* at 817-18; *id.* at 821 (Boudin, J., concurring) (discussing user lock-in).

[136] Macros are essentially mini-computer programs that allow a user to run a software

would again strongly discourage them from switching.[137] To combat the lock-in problem, Borland included the Lotus command hierarchy in its menus as an alternative interface (but just as with Google's Android, the functional computer code behind the command hierarchy was completely different).[138]

The First Circuit analyzed the issue under § 102(b) of the Copyright Act:

> In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.[139]

The court agreed with Borland that Lotus 1-2-3's menu structure was a "method of operation" and thus uncopyrightable under § 102(b).[140] "The Lotus menu command hierarchy does not merely explain and present Lotus 1–2–3's functional capabilities to the user; it also serves as the method by which the program is operated and controlled."[141] As a consequence, Borland needed to be able to use it to operate its programs in substantially the same way.[142]

The *Lotus* court recognized several important features of software interfaces that should have guided the Federal Circuit in its analysis. First, on the distinction between an interface and its underlying code, the court observed:

> The Lotus menu command hierarchy is also different from the underlying computer code, because while code is necessary for the program to work, its precise formulation is not. In other words, to offer the same capabilities as Lotus 1–2–3, Borland did not have to copy Lotus's underlying code (and indeed it did not); to allow users to operate its programs in substantially the same way, however, Borland had to copy the Lotus menu command hierarchy.[143]

---

application's commands in a particular order, thus permitting the user to automate repetitive tasks. Macros can be quite complex and thus difficult to rewrite for a new application that can perform the same commands but uses different names for them. *See id.* at 809-10 (discussing macros).

[137] *See id.* at 818 (discussing the difficulties in requiring users to rewrite macros).

[138] *See id.* at 810 (noting that Quattro contained "a virtually identical copy of the entire 1-2-3 menu tree," but "Borland did not copy any of Lotus's underlying computer code; it copied only the words and structure of Lotus's menu command hierarchy").

[139] 17 U.S.C. § 102(b).

[140] *See Lotus*, 49 F.3d at 815.

[141] *Id.*

[142] *See id.* at 816.

[143] *Id.* at 816.

Further, the *Lotus* court was much less impressed with arguments about the 'creativity' involved in structuring the menu hierarchy, recognizing the limited role such an analysis plays in the analysis of the functional aspects of computer programs:

> Accepting the district court's finding that the Lotus developers made some expressive choices in choosing and arranging the Lotus command terms, we nonetheless hold that that expression is not copyrightable because it is part of Lotus 1–2–3's "method of operation." . . . If specific words are essential to operating something, then they are part of a "method of operation" and, as such, are unprotectable. . . .
>
> The fact that Lotus developers could have designed the Lotus menu command hierarchy differently is immaterial to the question of whether it is a "method of operation." . . . Concluding, as we do, that users operate Lotus 1–2–3 by using the Lotus menu command hierarchy, and that the entire Lotus menu command hierarchy is essential to operating Lotus 1–2–3, we do not inquire further whether that method of operation could have been designed differently. The "expressive" choices of what to name the command terms and how to arrange them do not magically change the uncopyrightable menu command hierarchy into copyrightable subject matter.[144]

Finally, the court recognized the unique nature of software development:

We also note that in most contexts, there is no need to "build" upon other people's expression, for the ideas conveyed by that expression can be conveyed by someone else without copying the first author's expression. In the context of methods of operation, however, "building" requires the use of the precise method of operation already employed; otherwise, "building" would require dismantling, too. Original developers are not the only people entitled to build on the methods of operation they create; anyone can. Thus, Borland may build on the method of operation that Lotus designed and may use the Lotus menu command hierarchy in doing so.[145]

Even more pointed was Judge Boudin's concurrence, which was explicitly concerned with protecting the investment of *users* in learning the system and writing macros for it.[146] Competitors would have no way of competing

---

[144] *Id.*

[145] *Id.* at 818 (footnote omitted).

[146] *See id.* at 819-22 (Boudin, J., concurring).

with Lotus's installed user base unless they could take advantage of the users' investments in learning the 1-2-3 interface and writing macros for it; the users would instead be locked in to using Lotus 1-2-3.[147] Judge Boudin therefore concurred with the majority's holding that the menu hierarchy was an unprotectable 'method of operation' as the best available option for preventing Lotus from capturing these user investments for itself, and thereby holding users captive to its inferior product.[148]

That same cogent analysis that the First Circuit adopted in *Lotus* applies to Android: Google used the Java API declarations to take advantage of programmers' investments in learning the language, in the course of doing something different and (arguably) better. The district court in *Oracle v. Google* sensibly relied in part on *Lotus* to find in favor of Google.[149] Unfortunately, the Federal Circuit rejected this analysis, erroneously concluding that "the district court's reliance on *Lotus* is misplaced because it is distinguishable on its facts and is inconsistent with Ninth Circuit law."[150]

The AAI Brief cites *Lotus* (and particularly its concurrence) frequently, and much of its antitrust analysis is couched in similar terms. It notes that both cases provide examples of 'network effects.'[151] "Network effects arise when a service's value increases along with its number of users."[152] Network effects lead to a phenomenon called 'lock-in,' where users become locked into a particular system simply because they and everyone in their network have invested so much into that system.[153] This principle applies strongly in computer markets, especially for operating systems: As more people use a

---

[147] *See id.* at 821 (Boudin, J., concurring).

[148] *See id.* at 821-22 (Boudin, J., concurring).

[149] *See Oracle I*, 750 F.3d at 1364 (citing *Oracle America, Inc. v. Google LLC*, 872 F. Supp. 2d 974, 976-77 (N.D. Cal. 2012)).

[150] *Id.* at 1365; *see id.* at 1364-68 (distinguishing *Lotus v. Borland*).

[151] *See* AAI Amicus Brief, *supra* note 2, at 7-12 (section entitled "Affording Copyright Protection to Software Interfaces Can Entrench Dominant Firms Protected by Network Effects"); *see also* Computer Scientists Amicus Brief, *supra* note 9, at 22-23 (discussing network effects).

[152] Computer Scientists Amicus Brief, *supra* note 9, at 22. And the effect is not linear but exponential. *See* AAI Amicus Brief, *supra* note 2, at 9 ("According to 'Metcalfe's Law,' the proportional value to a network of a user's investment in joining the network is the square of the number of users who do so, such that a tenfold increase in the size of the network leads to a hundredfold increase in its value." (internal quotations omitted)).

[153] *See* AAI Amicus Brief, *supra* note 2, at 9-11 (section entitled "Lock-In Can Cement Software-Based Monopolies"); Computer Scientists Amicus Brief, *supra* note 9, at 24-25 (discussing lock-in).

given system, more software gets written for it, which in turn makes more people want to use it, etc.[154] As a general rule, these network effects swamp out any other effects and are very hard for competitors to overcome, as convincing locked-in users to incur the costs of switching is difficult, even if the new system is superior.[155] The *Lotus* court's decision was its attempt to avoid this lock-in effect,[156] and now amici are urging the Supreme Court to do the same.

For example, after describing the recognized problems that occur when patents cover industry standards, the AAI Brief states:

> A similar problem arises with copyrighted software interfaces. Copyright on largely functional elements of software that become an industry standard gives a copyright holder anticompetitive power to thwart or tax innovative developments that build upon the elements, and to misappropriate for itself investments by users or developers in learning those elements.[157]

Later, it adds

> Software-based markets are characterized by strong positive network effects, which means lock-in increases over time because switching costs increase as the network size increases and network participants make greater investments in training to learn the system. New entrants seeking to introduce a rival operating system must overcome the costs of inducing *both* consumers and software developers (as well as complementors) to switch to the new network.[158]

---

[154] *See* AAI Amicus Brief, *supra* note 2, at 9 (noting that "In an operating system environment, both consumers and software developers (as well as hardware and other complementors) invest in learning system software, adding several different dimensions of value to the network.").

[155] *See id.* at 9 ("Software-based markets are characterized by strong positive network effects, which means lock-in increases over time because switching costs increase as the network size increases and network participants make greater investments in training to learn the system."); Computer Scientists Amicus Brief, *supra* note 9, at 22-23 ("They make users unlikely to switch to technically 'better' competing software services that have not yet established a large userbase because much of a service's value comes from its community of users and its secondary market of compatible services.").

[156] *See Lotus*, 49 F.3d at 821 (Boudin, J., concurring) ("If Lotus is granted a monopoly on this pattern [of menu items], users who have learned the command structure of Lotus 1–2–3 or devised their own macros are locked into Lotus.").

[157] AAI Amicus Brief, *supra* note 2, at 7-8.

[158] *Id.* at 9-10 (internal citations omitted).

The AAI Brief cites the common example of the ubiquitous QWERTY keyboard, which is incredibly inefficient (and indeed was designed that way), but it remains entrenched because lock-in and network effects make switching incredibly costly and difficult for any single person, or even an organization.[159]

After laying out its antitrust case for *why* software interfaces like the API declarations should not be protected by copyright, the AAI Brief goes on to make its case for *how* to deny protection to such declarations,[160] largely along the lines outlined above. It is particularly critical of the Federal Circuit for failing to consider the balance between providing copyright protection and preserving competition, as necessitated by the Constitution's intellectual property clause:

To achieve this balance in practice, Congress and the courts have introduced a variety of competition safeguards that limit the scope of copyright protection, including § 102(b) of the Copyright Act, merger, and fair use. But instead of ensuring that the anticompetitive harm threatened by copyright protection of software interfaces is cabined by a sufficiently liberal reading of these limiting doctrines, the Federal Circuit relegated each of them to insignificance.[161]

The brief particularly faults the Federal Circuit's analysis for dismissing concerns about interoperability across software platforms as irrelevant throughout its opinions. Instead, the AAI Brief argues, the court's central focus should have been deploying copyright's limiting doctrines to provide robust protection of competitors seeking to enhance interoperability, as it is the best (and perhaps only) way to avoid lock-in and promote competition in the software industry.[162] The AAI Brief also backs up its interoperability analysis by liberally citing from a recent report from the Copyright Office,

---

[159] *See id.* at 4 ("Better typewriter keyboard layouts may exist, but the familiar QWERTY keyboard dominates the market because that is what everyone has learned to use." (quoting *Lotus*, 49 F.3d at 819-20 (Boudin, J., concurring)); *id.* at 10 (noting that the "inefficient QWERTY keyboard layout persists because 'the human component of the system' raises collective switching costs and creates significant difficulties for coordinating a move to superior technology" (quoting Carl Shapiro & Hal R. Varian, Information Rules 184, 185-86 (1999))).

[160] *See id.* at 12-22 (sections entitled "Copyright's Limiting Doctrines Are Designed to Preserve Innovators' Ability to Create Interoperable and Competitive Products and Services" and "The Court Should Sanction Utilitarian Transformations of Software Interfaces as Fair Use").

[161] *Id.* at 12.

[162] *See, e.g.*, *id.* at 12-19 (section entitled "Copyright's Limiting Doctrines Are Designed to Preserve Innovators' Ability to Create Interoperable and Competitive Products and Services").

Software-Enabled Consumer Products, which directly addresses the issue of the scope of software copyrights.[163] The AAI Brief invokes interoperability as a facet of the idea-expression dichotomy, as noted by the Copyright Office Report.[164] It also quotes the Report in its discussion of how interoperability fits with merger: "The Copyright Office also observed that the doctrine of merger is 'a promising avenue to permit copying for purposes of interoperability, at least in the narrow circumstances in which [it] appl[ies].'"[165] And the AAI Brief is quite forceful when it discusses interoperability as a significant factor that should weigh heavily in favor of fair use, again relying on the Copyright Office Report.[166] The AAI Brief concludes with: "This Court should recognize an appropriate role for copyright's limiting doctrines to promote competition and consumer welfare in software markets by allowing innovators to use software interfaces to create interoperable products and services."[167]

As argued by the various amici, the Federal Circuit in its decisions in *Oracle v. Google* fundamentally misunderstood the nature of software and software copyright law. The court failed to recognize the special nature of software, instead heedlessly applying precedents from other types of works that simply do not fit software (and disregarding applicable software precedents). It also failed to apprehend the ways in which its decision will fundamentally

---

[163] U.S. Copyright Office, Software-Enabled Consumer Products (Dec. 2016), *available at* https://www.copyright.gov/policy/software/software-full-report.pdf). This Report in large part takes the same analytical approach to software copyrights advocated by the AAI Amicus Brief.

[164] *See* AAI Amicus Brief, *supra* note 2, at 14 ("[T]he fact that copying a software interface is necessary to achieve interoperability or compatibility, and that a defendant has copied no more than is necessary to do so, tends to prove that a plaintiff's infringement claim falls on the wrong side of the idea/expression dichotomy . . . ."); *id.* at 15 (quoting Software-Enabled Consumer Products, *supra* note 163, at 52, as recognizing "the importance of preserving the ability to develop products and services that can interoperate with software-enabled consumer products, and the goal of preserving competition in the marketplace," but that further legislation was not needed because "faithful application of existing copyright law doctrines can preserve the twin principles of interoperability and competition").

[165] *Id.* at 16 (quoting Software-Enabled Consumer Products, *supra* note 163, at 54).

[166] *See id.* at 17-18 ("The Copyright Office further 'believes that, in many cases, copying of appropriately limited amounts of code from one software-enabled product into a competitive one for purposes of compatibility and interoperability should also be found to be a fair use.'" (quoting Software-Enabled Consumer Products, *supra* note 163, at 57)); *see also id.* at 18 (noting that the Federal Circuit recognized the importance of interoperability to the assessment of fair use in its copyright analysis in *Oracle I*, but then dismissed those considerations in its actual fair use analysis in *Oracle II*); *id.* at 21 (noting that, in terms of the fair use analysis, the Report observes that "'interoperability is a favored use'" (quoting Software-Enabled Consumer Products, *supra* note 163, at 57).

[167] *Id.* at 22.

harm competition and innovation in the software industry, to the detriment of consumers everywhere. The Supreme Court must therefore reverse the Federal Circuit and restore competitive balance to the software indust