Summer 1994

Right to Reverse Engineer Software: Is Japan Next and Does It Really Matter

John Espenshade Titus

Follow this and additional works at: http://scholarship.law.unc.edu/ncilj

Recommended Citation

This Comments is brought to you for free and open access by Carolina Law Scholarship Repository. It has been accepted for inclusion in North Carolina Journal of International Law and Commercial Regulation by an authorized editor of Carolina Law Scholarship Repository. For more information, please contact law_repository@unc.edu.
Right to Reverse Engineer Software: Is Japan Next and Does It Really Matter

Cover Page Footnote
International Law; Commercial Law; Law

This comments is available in North Carolina Journal of International Law and Commercial Regulation:
http://scholarship.law.unc.edu/ncilj/vol19/iss3/4
Right to Reverse Engineer Software: Is Japan Next and Does It Really Matter?†

I. Introduction

Although computer programmers may see software as a form of art¹ and worthy of high levels of protection,² copyright laws around the world are converging on a more pedestrian viewpoint.³ Following the lead of the United States⁴ and the European Community,⁵ Japan is considering delimiting the rights of software producers by allowing purchasers to reverse engineer computer programs.⁶

The proposal has serious ramifications for bilateral trade. Software is one of the few industries in which the United States is an international colossus, accounting for seventy-eight percent of worldwide sales⁷ and fifty-five percent of sales in Japan.⁸ The success of most foreign products in Japanese markets is dismal by comparison.⁹ One of the reasons for the success of U.S. software producers in Japan has

† This Comment has been entered in the 1994 Nathan Burkan Memorial Competition.
² See generally CLAPES, infra note 1 (arguing that the non-literal aspects of computer software deserve higher protection than that afforded in recent decisions); Note, Copyright Law—Scope of Protection of Non-Literal Elements of Computer Programs—Second Circuit Applies "Abstraction-Filtration-Comparison" Test, 106 HARV. L. REV. 510, 511 (1992) (arguing that the abstraction-filtration-comparison "test will discourage innovative programming techniques and leave non-literal elements of computer programs under-protected.").
³ See infra notes 37-43 and accompanying text for a discussion of the nature of software.
⁴ See infra notes 44-138 and accompanying text for a discussion of U.S. law.
⁵ See infra notes 139-59 and accompanying text for a discussion of European Community law.
⁶ See infra Part II for a discussion of the Japanese proposal and the U.S. response to it.
⁷ See Negotiators Make No Progress in Intellectual Property Talks, 11 Int'l Trade Rep. (BNA) No. 1, at 10 (Jan. 5, 1994) [hereinafter Negotiators]. By itself, Microsoft has achieved incredible success. Worldwide there are 140 million personal computers, 90% of which use either Microsoft's DOS or Windows operating systems. See Mark II, ECONOMIST, Mar. 19, 1994, at 81.
⁹ Foreign (non-Japanese) penetration of Japanese markets hovers around 20% in semiconductors. Id. In telecommunications the figure is 5%, while for insurance it is 2%. U.S.-Japan Trade Talks Making Progress, USTR Official Says, 11 Int'l Trade Rep. (BNA) No. 50, at 2142 (Dec. 22, 1993) [hereinafter Trade Talks].
been the latter's failure to produce good software. Thus, in the words of one U.S. software industry official, the right of reverse engineering might mean that Japan "could instantly produce software that they have not had the ability to write themselves."

Moreover, the proposal arrives amidst the continuing problem of worldwide infringement of U.S. copyrights on software, which industry representatives say amounts to billions of dollars in losses worldwide, including three billion dollars per year in Japan. Compared to the total sales of the biggest U.S. software producers, such losses are massive. In 1993 IBM posted software sales of almost three billion dollars, while Microsoft’s sales were closer to four billion dollars.

Perhaps the best reason for concern is what the timing of the proposal may signal to American software companies. Japan is currently wallowing in its worst recession in fifty years. Faced with recessions in the past, Japan often increased its exporting efforts to lift its economy out of a morass. With the global demand for software so high, and Japan’s market share so low, a burst of software production by Japan would give the Japanese economy, in theory, a positive jolt. A broad right to reverse engineer computer programs could give Japanese programmers inspiring insight into the world’s most successful software. Yet the position that various members of the Clinton Administration have urged Japan to adopt is inconsistent with current

---


11 T.R. Reid & Peter Behr, A Software Fight's Blurred Battle Lines; U.S. Computer Companies Are on Both Sides as Japan Considers Copyright Law Changes, WASH. POST, Jan. 11, 1994, at D1 (quoting Marshall C. Phelps, Jr., Vice President of IBM).

12 See id. The rate of piracy, the ratio of illegal copies to total copies, is claimed to be 92% in Japan. Id. Other notorious infringers include Korea (yearly losses of $315 million, 88% piracy rate), Korea to Step Up Protection to Avoid Special 301 Designation, Official Says, 10 Int'l Trade Rep. (BNA) No. 13, at 528 (Mar. 31, 1993), France ($1.2 billion losses, 73% piracy rate), and the United States ($1.9 billion losses). See Software Firms Lost $1.1 Billion Due to Illegal Copying, Group Says, Int'l Trade Daily (BNA), June 3, 1993, available in LEXIS, Intlaw Library, BNAITD File.

13 See supra text accompanying notes 42-43.
U.S. law on reverse engineering.  

This Comment discusses the context of the Japanese proposal and the response it provoked within both the industry and the Clinton Administration in Part II. In Part III, the Comment briefly describes some programming concepts and the scope of reverse engineering. The law of both the United States and the European Community (EC) is discussed in Parts IV and V. In Part VI, the Comment concludes that the Japanese proposal is far less of a threat to the U.S. software industry than the domestic threat of overly broad applications of various U.S. copyright doctrines to infringement cases.

II. Japanese Proposal; U.S. Response

On July 22, 1993, the Japanese Agency of Cultural Affairs announced that it was studying the possibility of amending Japanese copyright law to allow software users to reverse engineer or decompile computer programs. The proposal came less than two weeks after the United States and Japan announced plans for future negotiations over measures to reduce the U.S. trade deficit with Japan, which topped forty-nine billion dollars in 1992.

The proposal represented something of a reversal of the Japanese stance on computer software. Weeks prior to the reverse engineering initiative, the Japanese Agency of Cultural Affairs had proposed increased restrictions on the use of computer software and greater enforcement of a law forbidding the removal of copy protection schemes.

Despite the potential impact of the proposed measure, the response of U.S. government and industry was belated. In November 1993, Commerce Secretary Ron Brown and U.S. Trade Representative

---

18 See infra Parts II and IV.B. But see Bob Rossi, A Brash, New Voice for the IP Arena, Recorder, Feb. 17, 1994, at 1 (Patent and Trademark Commissioner Bruce Lehman "has maintained that U.S. copyright law prohibits [reverse engineering] under all but a few circumstances."). The U.S. Assistant Commissioner for External Affairs at the Patent and Trademark Office, Michael Kirk, on the other hand, says that the only two U.S. cases to address the issue of reverse engineering "were exceptions." See Andrew Pollack, U.S. Protesting Japan's Plan to Revise Software Protection, N.Y. Times, Nov. 22, 1993, at D2.

19 See Negotiators, supra note 7, at 10.


One year later, the trade deficit had climbed to around $60 billion, worsening trade relations between the two countries. See Ruth Marcus & Peter Behr, U.S., Japan Talks on Trade Collapse; Clinton Silent on Possible Sanctions, Wash. Post, Feb. 12, 1994, at A1.

21 See Japan Government Weighs Restrictions on Software Copying by Individuals, 10 Int'l Trade Rep. (BNA) No. 27, at 1128 (July 7, 1993).

22 According to Robert Holleyman, president of the Business Software Alliance, which includes industry powerhouses such as IBM, Microsoft, Lotus, and Apple, the United States did not learn of the proposal until September 1993. See Barshefsky, supra note 8. In July 1993, however, the proposal was announced in the Japanese equivalent of the Wall Street Journal, Nihon Keizai Shimbun. See Robert Patton, Software, Sci. Am., Apr. 1994, at 116. According to another source, IBM first learned of the proposal informally and alerted other U.S. concerns. See Reid & Behr, supra note 11, at D1.
Mickey Kantor complained in a letter to the Japanese Trade Minister that the Agency's proposal conflicted with the spirit of the July trade negotiations. In early December, Charlene Barshefsky broached the matter in Tokyo at the ongoing bilateral negotiations on intellectual property, while at an even higher level, U.S. Ambassador Walter F. Mondale voiced his opposition to the proposal directly to Japanese Education Minister Ryoko Akamatsu.

Unlike the Clinton Administration, the U.S. computer industry is split on the matter. The Business Software Alliance (BSA), like the U.S. government, opposes the Japanese proposal, while the American Committee for Interoperable Systems (ACIS) favors it. ACIS comprises thirty companies whose primary function in the computer industry is the manufacture of hardware. ACIS contends that the right to reverse engineer will permit hardware manufacturers to achieve compatibility with software more easily. Under the current regime in Japan, ACIS says, Japanese hardware makers have closed American companies from the market by refusing to divulge the hardware specifications necessary to make Japanese software run properly.

BSA, on the other hand, comprises software producers, who worry that if Japanese software makers have free access to the ideas underlying American software, they will be able to produce competing software at a lower price than American firms.

The U.S. government has taken the side of BSA and has pressured Japan repeatedly not to adopt the reverse engineering measure. On December 13, 1993, the Japanese Agency of Cultural Affairs held a hearing on the matter. Testifying were members of the U.S. software industry and officials from both the Patent and Trademark Office and the Department of Commerce. The hearing occurred amidst the "economic framework talks" between the United States and Japan. While the general trade talks appeared successful, talks on intellectual property went nowhere. These negotiations were followed in February 1994 by the U.S.-Japanese Economic Summit, which was also
The reverse engineering issue, of great importance to the United States and on which Japan is presently silent, will most likely surface again by the July 1994 meeting of the G-7.\textsuperscript{36} The soundness of the U.S. position is discussed in Part VI.

III. Computer Programs and Decompilation

The law of reverse engineering of computer software is most often cast in terms of “decompilation.”\textsuperscript{37} Decompilation, however, is only one form of reverse engineering. Generally, there are several levels of computer programming. Before anything happens, the programmer conceives of a task. To accomplish this task, he writes an algorithm. An algorithm is the most abstract level of programming and may exist only on paper or even in the programmer’s head. To average a batch of numbers, for example, the computer must add the numbers, then divide the total by the number of numbers. The algorithm for this simple task may look like this:

\begin{verbatim}
Begin program
Count = 0; total = 0
Input no. to computer
Add no. to total
Add 1 to count
Another no.? Yes
No.
Avg. = Total/Count
\end{verbatim}

The trick in programming is to convert the algorithm into a running program. The brains of the computer lie in the microprocessor, a large chip that operates on electrical impulses. Each impulse is either a zero or a one, which equates to a transistor state of either off or on. Each piece of data that comes through the microprocessor must come as a stream of zeros and ones. Therefore, before the algorithm in the above diagram can be implemented by the computer, it must be translated into a form recognized by the microprocessor.

First, the programmer must convert the algorithm into source code. Source code is less intelligible to programmers than an algorithm, but is fairly readable nevertheless.\textsuperscript{38} Source code, like human speech, comes in languages, such as BASIC, C, or Pascal. Unlike

\textsuperscript{35} See Marcus & Behr, \textit{supra} note 20, at A1.
\textsuperscript{36} See \textit{Special Report}, \textit{supra} note 16, at 105.
\textsuperscript{37} In \textit{Sega Enters. v. Accolade}, 977 F.2d 1510 (9th Cir. 1992), defendant used plaintiff’s computer program, which was in object code, to generate source code. \textit{Id.} at 1514-15. The \textit{Sega} court blurs the distinction between decompilation and reverse engineering, \textit{id.} at 1514, and between compiling and assembling, \textit{id.} at 1514 n.2.
\textsuperscript{38} In \textit{Pascal}, the algorithm above may be implemented as follows:
human speech, however, the lexicon of a particular program is really just an instruction set that is severely limited by syntax.

The program may next be assembled.\(^3^9\) Assembled source code is written in assembly language. The vocabulary of assembly language is even more limited than that in source code, and the instructions are shorter. The form of the program, moreover, is ultimately dictated by the constraints of the microprocessor, such as its memory size and location, instruction set, and protocol. Although alphanumeric, assembly language is for the most part indecipherable.\(^4^0\) Note that assembly code is not yet intelligible to the computer, which can read only ones and zeros.

Finally, the program must be compiled. Compiling entails converting the assembled program into a stream of ones and zeros. Conceptually, this phase is not difficult. Each assembly command, e.g., MOV, is assigned a number that is expressed in binary. (The binary representation of the number 13, for example, is 1101.) The ADD command may be assigned 0101, while AX (a data register in the microprocessor's memory) is 10 and BX is 01. Thus, the command "ADD AX, BX" is compiled as 01011001. This is object code, also known as machine code because it is intelligible to the computer.

To disassemble a program is to convert the compiled object code

```plaintext
program Average;
Var
  count : integer;
  ans : char;
  N, avg, total : real;
begin
  count : = 0;
  total : = 0;
  repeat
    Write ('Enter a number: ');
    Readln (N);
    count : = count + 1;
    total : = total + N;
    avg : = total / count;
    Write ('Another number? (y or n)');
    Readln (ans)
    until (ans = 'n') or (ans = 'N');
  Writeln ('The average is ', avg)
end.
```

\(^3^9\) A program need not be assembled. Assembly language is mentioned because disassembly—converting object code into assembly language—is easier than decompiling—converting object code into source code. When speed of execution is necessary, programmers often turn to assembly language. See John Norton & Phillip Socha, Assembly Language for the IBM PC xiii-xiv (1986) ("Typical assembly language programs are two to three times as fast as equivalent C or Pascal programs.").

\(^4^0\) An assembled program is generally a long set of steps that might appear as follows:

```plaintext
MOV DX, SS: [BX + 2]
XOR BX, BX
MOV CX, Count
JCXZ SD4
CMP Data [BX], AX
JL SD2
CMP Data [BX], DX
```
into assembly language, which is more readable than just ones and zeros. Decompiling entails converting a program from object code into source code. Disassembling and decompiling are both forms of reverse engineering software, as they each offer insight into the ideas that underlie the object code. The distinction is often ignored in discussions of copyright, but it is helpful conceptually.41

When commercial software vendors release programs, they often release only the object code (ones and zeros). The reason is fairly straightforward. The computer understands and operates only on object code, so the source code is not necessary to the physical execution of the software. As far as the vendor is concerned, releasing the source code is not only superfluous, but dangerous. Often programmers have devised tricks, shortcuts, or innovative subroutines that increase the efficiency of the program or make it more pleasing to the eye. Also, the software may contain data in the nature of trade secrets, such as empirically determined parameters or constants.42 Innovations of this sort make the software more marketable, but come at a cost. A competitor who learns these secrets could market the same software at a lower price. It is not surprising that software producers want to shield these secrets. If the source code is included with the object code, these secrets will be transparent to a user who knows what he is looking for in the code. But when such information is buried deep within an endless stream of object code, it is effectively encrypted.43

IV. American Law

In the United States, the judicial development of copyright protection for computer software began in the late 1970s.44 After settling on

41 See supra note 37.
42 See, e.g., Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823 (10th Cir. 1993).
45 In 1980, Congress added section 117, which recognizes two realities of software. First, for the purchaser of software to run it on a computer, he must make a copy of it in the computer's random access memory (RAM). Second, computer glitches sometimes erase the program itself, so a backup copy is a necessary form of insurance. The right to make a copy of a program for these two purposes was thus granted. See 17 U.S.C. § 117 (1988). Without these rights, the mere use of the program or making of a backup copy would infringe the right of reproduction. See 17 U.S.C. § 106(1) (1988).

Congress also responded to abuses. In copyright law, the first sale doctrine limits the copyright holder's distribution right, 17 U.S.C. § 106(3), to the first sale, after which the owner of the physical copy may dispose of the copy at his pleasure, whether by loan, resale, or rental. A software rental industry soon mushroomed. A rental outfit would purchase the software from the copyright owner, say IBM. It would then rent the software to patrons for
the view that computer programs may be copyrighted whether in the form of source code or object code, courts next wrestled with the appropriate method of proving infringement.

Absent direct proof of copying, which is rarely the case, the standard formula for infringement is access plus substantial similarity. The second method of proof is inferential. If the defendant had access to the plaintiff’s work, and the allegedly infringing work is substantially similar to the first, the court (or jury) may find that copyright infringement has occurred—the defendant copied the plaintiff’s work and violated his exclusive right of reproduction. As access to mass marketed software is fairly easy to prove, the courts were left to decide what constitutes substantial similarity.

Recently, U.S. courts have begun to trim back the copyright protection of computer software. The motivating force behind this development is the distinction between ideas, which are not protectable, and expression, which is. Although the distinction is impossible to formulate definitively, Judge Learned Hand put forth a classical construct to visualize the range between idea and expression in *Nichols v. Universal Pictures Corp.* Lying between expression and idea, Hand

---

one or two nights for a fraction of the actual purchase price. As anyone who has worked with it knows, commercial software cannot be mastered in a night, so the likely purpose of the rental was to copy the software. The copyright holders had no cause of action against the rental outfits, which were exercising their rights under the first sale doctrine. Congress thus carved out an exception to the doctrine, prohibiting the owners or possessors of software from "rental, lease, or lending" of it. See 17 U.S.C. § 109(b)(1)(A) (Supp. III 1991).

45 See Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983); Stern Elecs., Inc. v. Kaufman, 669 F.2d 852 (2d Cir. 1982); Williams Elecs., Inc. v. Artic Int’l, Inc., 685 F.2d 870 (3d Cir. 1982).

The point seems obvious now, but at the time the debate was alive, many good arguments were advanced against copyrighting computer programs at all. See, e.g., Pamela Samuelson, *CONTU Revisited: The Case Against Copyright Protection for Computer Programs in Machine-Readable Form*, 1984 Duke L.J. 663. One of the staple arguments was that "bringing new ideas into the public domain was the quid pro quo the public received in the exchange for the limited monopoly right the author received to protect his or her expression of the ideas." *Id.* at 705-06.

Arguments on the subject, of course, date back to the time when computers were in their infancy. See, e.g., Stephen P. Breyer, *The Uneasy Case for Copyright: A Study of Copyright in Books, Photocopies, and Computer Programs*, 84 Harv. L. Rev. 281, 340-50 (1970).


47 See, e.g., Kepner-Tregoe, Inc. v. Leadership Software, Inc. 12 F.3d 527, 532 (5th Cir. 1994).

48 See, e.g., Wildlife Express Corp. v. Carol Wright Sales, Inc., 8 F.3d 502 (7th Cir. 1994).

49 See, e.g., Whelan Assoc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222 (3d Cir. 1986) (applying Baker v. Selden, 101 U.S. 99 (1879), directly to computer software and concluding that a program’s overall function was its idea and that its structure, sequence, and organization were protectable expression). For a brief discussion of *Baker*, see *infra* notes 58-60.


said, is a series of abstractions "of increasing generality." Thus, the text of a play is copyrightable, while its plot is not. The challenge in software cases has been to apply the abstractions test to computer code. Designed to carry out a programmer's idea in the circuitry of a computer, source code is an inevitable blend of idea and expression; all of it must be written not only with the computer specifications in mind, but with the goal of efficiency.

Copyright law has many doctrines that limit the protection of externally driven expression. The principle that expression, but not an idea, is protectable emerged in this setting. The plaintiff in *Baker v. Selden* developed a method of implementing double-entry bookkeeping on one or two pages. The defendant copied this system and marketed his own book of accounting forms, which differed from those shown by the plaintiff as examples. The U.S. Supreme Court held for the defendant, reasoning that the plaintiff could not protect the expression, (accounting forms) necessarily incidental to his accounting system, which was not protectable by copyright. The *Baker* holding, that ideas and systems may not be copyrighted, has since been codified in U.S. law.

A related doctrine is that of merger, which holds that when an idea may be expressed in a limited number of ways, the expression merges into the idea and is not protectable. In *Morrissey v. Proctor & Gamble Co.*, the First Circuit Court of Appeals held that sweepstakes rules on a card were of necessity so brief as to permit only a limited number of expressions and thus were not copyrightable. The effect of allowing copyright protection in such cases would be to grant a de

---

53 Id. at 121.
54 Id.
55 In a version of Pascal called Turbo Pascal 4.0, for example, the largest value an integer variable can take is 32,767. *Turbo Pascal Owner’s Manual* (Borland ed., 1987).
56 Code is efficient to the extent that it accomplishes a given task with the fewest bits (ones and zeros) of object code. The notion that efficiency is the preeminent goal of programming has been challenged lately. Many programmers believe that because programs must be updated continually as new versions of software are released, clarity is an equally important goal. See, e.g., Sid L. Huff, *Object Oriented Programming; Info Tech, Bus. Q.*, Dec. 22, 1993, at 85.
57 The concept of externally driven expression may be illuminated by comparing the author who writes free form poetry with the one who must write the instructions for affixing a stamp to an envelope. In the latter case, many real world or physical considerations will dictate the expression. For example, one cannot apply pressure to the stamp before it has been moistened. If a second author also writes instructions for putting a stamp on an envelope, certain similarities will result as a matter of course.
59 Id. at 104-05.
60 See 17 U.S.C. § 102(b) (1988). "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." Id.
61 See, e.g., Herbert Rosenthal Jewelry Corp. v. Kalpakian, 446 F.2d 738 (9th Cir. 1971) (idea of a jeweled bee pin held to have merged with its expression).
62 379 F.2d 675 (1st Cir. 1967).
facto monopoly on the idea. The doctrine often determines the outcome of cases where instructions or linguistic economy drive the expression. Both settings are applicable to computer code.

The *scenes a faire* doctrine governs shopworn conventions, which fail the originality standard. Conventions evolve in many settings. Fairy tales begin with "Once upon a time." College profiles invariably contain information about tuition, fees, majors, entrance exam scores, housing, and meal plans. When these conventions first originate, of course, they are original. Over time, however, they become hackneyed and are no longer copyrightable. The computer science culture has exploded in a short period of time, and the *scenes a faire* doctrine is now commonplace in computer software cases. With thousands of computer scientists writing programs in the most efficient manner possible for a relatively small number of computer systems, the convergence to certain conventions was perhaps inevitable.

*Scenes a faire* may limit copyright protection in a second setting that is similar, yet seemingly distinct. A roguish knight may be a stock character in a play about the Middle Ages, but his inclusion is not necessary to the work. This is not true for many pieces of software written to perform a specific task. Here the inclusion of certain data or

---

63 See id.
64 Compare Financial Info., Inc. v. Moody's Investors Serv., 808 F.2d 204 (2d Cir. 1986), cert. denied, 484 U.S. 890 (1987) (holding that as a compilation, a card with five facts about an investment bond lacked sufficient originality for protection) with Kregos v. AP, 937 F.2d 700 (2d Cir. 1991) (holding that a pitching form with nine statistics about a starting pitcher was copyrightable subject matter).
65 The *scenes a faire* doctrine was introduced to American copyright law in Cain v. Universal Pictures Corp., 47 F. Supp. 1013 (S.D. Cal. 1942).
67 The *scenes a faire* doctrine originally had a more restricted meaning. From *Cain* until Reyher v. Children's Television Workshop, 533 F.2d 87 (2d Cir.), cert. denied, 429 U.S. 980 (1976), the doctrine withheld protection from expression that "necessarily results from the fact that the common idea is only capable of expression in more or less stereotyped form" by not including the expression in the substantial similarity analysis. *Id.* at 91 (emphasis added).
69 Learned Hand, however, saw the danger of withholding protection from a hackneyed cliche, which, though unoriginal in the sense that the idea is obvious, may be *expressed in myriad ways*. See Sheldon v. Metro-Goldwyn Picture Corp., 81 F.2d 49, 53-54 (2d Cir. 1936) ("[A]nticipation as such cannot invalidate a copyright.").
68 See, e.g., Data E. USA, Inc. v. Epix, Inc., 862 F.2d 204, 208 (9th Cir. 1988) (applying *scenes a faire* to a home computer video game).
69 Windows, pull-down menus, and icons are all examples of standards that have emerged.
70 Nichols v. Universal Pictures Corp., 45 F.2d 119 (2d Cir. 1930), cert. denied, 282 U.S. 902 (1931). "If Twelfth Night were copyrighted, it is quite possible that a second comer might so closely imitate Sir Toby Belch or Malvolio as to infringe, but it would not be enough that for one of his characters he cast a riotous knight who kept wassail to the discomfort of the household . . . ." *Id.* at 121.
subroutines may be mandatory. As a result, any program written to perform a specific task necessarily bears strong similarities to other such programs. At least one court has said that the *scenes a faire* doctrine bars protection in such instances, at least to the extent of protecting necessary elements in the program. This second setting is a subset of the first: an element that is common to a genre of work may not be necessary, but a necessary element will perforce be common. The second defense has been deemed the externalities defense.

Writing computer programs (or compiling databases) is labor intensive, so it is not surprising to find advocates for the sweat of the brow doctrine, which holds that the labor wrapped up in a work ought to be protected by copyright. The U.S. Supreme Court, however, has explicitly rejected this notion. Despite the array of names, these doctrines reflect only two basic copyright principles codified by U.S. law. First, a copyright extends protection to a work's expression; any ideas, processes, or systems contained in a work are not protectable. Second, expression must be original to earn copyright protection.

These principles are deceptively simple, and two points are worth noting. First, the Supreme Court's recent pronouncement that the originality standard is low but not zero is helpful only at the level of literal expression: the defendant in *Feist* made exact copies of the plaintiff's phone book. The Court looked to the nature of the work (an alphabetized list of names accompanied by addresses and phone numbers) and held that the compilation was so mechanical as to be void of any originality. The Court did not need to venture into the murky region of nonliteral expression, where originality is more difficult to define. The *scenes a faire* doctrine is one limit on the protection of nonliteral elements, but has not been developed fully.

---

71 *See Plains Cotton Coop. Ass'n v. Goodpasture Computer Serv., Inc.*, 807 F.2d 1256, 1262 (5th Cir.), cert. denied, 484 U.S. 821 (1987) (holding that software which analyzes cotton markets by necessity contains certain facets, which are thus not protectable).
72 *See id.* The court noted that "many of the similarities between [plaintiff's and defendant's] programs are dictated by the externalities of the cotton market" and were thus similar to unprotectable computer input formats. *Id.* at 1262 (citing Synercom Technology, Inc. v. University Computing Co., 462 F. Supp. 1003 (N.D. Tex. 1978)).
73 That is, the cost of the inputs is dominated by labor costs as opposed to the capital costs, which are comparatively minimal. This does not mean that market entry is easy, however. "It can cost hundreds of millions of dollars to develop a new computer operating system and get the product established on the market." *Computer Executives Maintain Industry Generally Enjoys Healthy Competition, Antitrust & Trade Reg. Rep. (BNA) No. 1635, at 490 (Oct. 14, 1993)* (quoting Paul Johnson, Vice President of the Equity Research Department of First Boston Corporation).
76 *Id.* § 102(b) (1988).
77 *Id.* § 102(a).
78 *See Feist*, 499 U.S. at 359.
79 *Id.*
The second (and related) point is that courts may use two legal devices to withhold protection from expression. They can say that the plaintiff's expression fails the two standards set by section 102 (no protection of ideas or systems, nor of unoriginal expression); or they can say that certain aspects of the defendant's work necessarily bear similarities to the plaintiff's work and are not dispositive of whether copying has occurred. Either way, these aspects of plaintiff's expression are left unprotected.

Though seminal, Judge Hand's abstractions test is inadeguatio in the face of so many programming considerations. In the absence of evidence proving actual copying, courts have allowed inferential showings of copying, achieved by demonstrating both access and substantial similarity between plaintiff's work and defendant's allegedly infringing work. Although the test seems easy to apply, wrinkles occur where two works contain similar or even identical nonprotectable elements. Where this is the case, substantial similarity of expression is irrelevant because the expression does not pass muster under section 102 and is thus not dispositive of copyright infringement.

Judge Hand's abstractions test is useful where expression is clear from the literal text, as is the case with a traditional literary work like a novel or play. The construct allows one to visualize a spectrum from pure expression to pure idea. At the level of pure expression is the verbatim copy of the play; at a level close to pure idea is its plot, e.g., a love story. Somewhere in between is the line of copyrightability. Problems arise here with software code. The literal code, i.e., the actual text of either the source code or the object code, is clearly protectable. But the nonliteral aspects of the code such as its structure, sequence, organization, order, and selection of subroutines, present difficulties. For a variety of reasons, some of these aspects are of dubious copyrightability because external constraints drive so much of computer programming, and often do so at a level that is very close to the literal expression in the code. The series of abstractions is short before one runs head on into the doctrines of merger and scenes a faire. But even if the line of originality hovers fairly close to the literal code, many programs stretch into the megabyte range, so that there is un-

---

80 The latter tack, of course, is not available where direct copying has been proved by plaintiff or conceded by defendant. See, e.g., id.; Sega Enters. v. Accolade, 977 F.2d 1510 (9th Cir. 1992).

81 Either it is unoriginal and fails under § 102(a), or it is of a nature forbidden by § 102(b), such as an idea, system, or procedure. See 17 U.S.C. § 102 (1988).

82 See supra notes 58-77 and accompanying text for a discussion of doctrines that raise hurdles establishing the copyrightability of program code.

83 A megabyte is one million bytes, or eight million bits. Most English words can be represented in eight bytes, so that one megabyte translates roughly into 125,000 words, about the length of a good-sized novel. See, e.g., Thomas A. Stewart, The Information Age in Charts, FORTUNE, Apr. 4, 1994, at 79 (A human genome, which requires 750 megabytes to describe, is tantamount to 399,000 pages of text). But recall that programs are driven by efficiency, so
doubtedly a good deal of protectable expression.

A. Computer Associates v. Altai

A workable solution to the weakness in the abstractions test was provided in *Computer Associates v. Altai*, which puts forth the abstraction-filtration-comparison (AFC) test. In the abstraction phase of the test, the computer program is visualized in the manner of Judge Hand's abstractions test: at the extreme end of expression lies the exact computer code; at the idea end of the program lies its overall purpose; and in between are the functions of each subroutine, the algorithms, data sets, and variable names. The necessary extension of Judge Hand’s test is the filtration phase, which winnows out expression that is unprotectable in light of several copyright doctrines. In the comparison phase, the remaining expression, which is protectable, is compared to that in the allegedly infringing program. A court then determines whether the defendant has copied this expression and weighs "the copied portion’s relative importance with respect to the plaintiff’s overall program." The protectable expression that remains after filtration may not be much. The court called this remainder a "kernel." Analogizing the filtration process to sifting for gold, one commentator noted that "[i]n many cases, the gold prospector will be disappointed, finding nothing but worthless sand and pebbles."

The abstraction-filtration-comparison test has gained quick acceptance.

---

that the analogy to a novel is not very strong. An encyclopedia of short plot summaries is perhaps a more accurate comparison.

---

84 982 F.2d 693 (2d Cir. 1992).
85 Computer Assocs., 982 F.2d at 707.
86 Id.
87 Id. at 702.
88 See supra notes 58-77 and accompanying text.
89 Computer Assocs., 982 F.2d at 710.
90 Id.
91 Id. at 706.
93 See Kepner-Tregoe, Inc. v. Leadership Software, Inc. 12 F.3d 527, 534 (5th Cir. 1994) (noting that the abstraction-filtration-comparison analysis of software has been "approved by courts for sophisticated treatment of copyright cases"); Gates Rubber Co. v. Bando Chem. Indus., Ltd., 9 F.3d 823, 834 (10th Cir. 1993) (noting that "[i]n substantial part, we adopt the ‘Abstraction-Filtration-Comparison’ test . . . ."; and remanding the case to district court to apply the filtration phase of the test in order to eliminate unprotectable elements, id. at 849); Autoskill v. National Educ. Support Sys., 994 F.2d 1476, 1490-91 (10th Cir. 1993) (analyzing substantial similarity using the abstraction-filtration-comparison test on an appeal from preliminary injunction, but not explicitly adopting it). But see Comprehensive Technologies Int'l, Inc. v. Software Artisans, Inc., 5 F.3d 750, 735 n.5 (4th Cir. 1993) (declining to decide whether to adopt abstraction-filtration-comparison for copyright infringement analysis). The test has made headway at the district court level as well. See, e.g., Apple Computer, Inc. v. Microsoft Corp., 821 F. Supp. 616, 625 (N.D. Cal. 1993) (applying filtration).
94 The most important American case on reverse engineering of computer software, *Sega Enters. v. Accolade*, 977 F.2d 1510 (1992), distinguishes *Computer Associates* on the ground
The AFC test applies only when two programs are compared in the process of determining whether one is substantially similar to the other. A second problem occurs when evidence shows that the defendant copied from the plaintiff's program directly, but limited copying appears in the defendant's program. This situation may arise where the defendant copies a program to learn of functional specifications or ideas that are not evident in the object code. Because such copying is not apparent in the defendant's final product, it is often called intermediate copying. Most of the Computer Associates case deals with this former problem: the court upheld the lower court's finding that defendant's Oscar 3.5 was not substantially similar to plaintiff's CA Scheduler. Evidence showed, however, that much of an earlier version of defendant's program had been taken directly from plaintiff's CA Scheduler. In this instance, the district court found that the defendant had infringed plaintiff's copyright and awarded the plaintiff over $360,000 in damages. The defendant did not contest the award on appeal.

B. Sega Enterprises v. Accolade

The case of direct copying seems straightforward, but is not for many of the same reasons that complicate the substantial similarity analysis. In Sega Enterprises v. Accolade, the defendant had written video game programs for use on Sega's game console. For a game to run on the console at all, however, the interface specifications of the console must be incorporated into the game program. Knowing that this information had to be embodied within Sega's own video games, Accolade's engineer's "wired a decompiler into the console circuitry, and generated printouts of the resulting source code." By that substantial similarity analysis is based "solely on the degree of similarity between the allegedly infringing work and the defendant's final product" and does not address the issue of intermediate copying that is central in Sega. See infra notes 101-38 and accompanying text.

Several cases deal with intermediate copying of standard literary works. See, e.g., Walker v. University Books, 602 F.2d 859 (9th Cir. 1979); Walt Disney Prods. v. Filmation Assocs., 628 F. Supp. 871 (C.D. Cal. 1986).

Interface specifications are information about the hardware on which the software is to run. For example, data may be sent from the software to the microprocessor over any number of channels (generally from eight to thirty-two). The number and location (addresses) of these channels must be incorporated into the software if it is to run properly.

Id. at 1515. For the operation of a decompiler, see supra text accompanying note 41.
exchanging and modifying the source code of Sega’s games, Accolade was able to learn the console specifications and incorporate them into its own games, which then ran on Sega’s console.\(^\text{106}\)

The reverse engineering process entailed making copies\(^\text{107}\) of Sega’s copyrighted video games,\(^\text{108}\) even though the final versions of Accolade’s games contained none of Sega’s code beyond the interface specifications.\(^\text{109}\) In its copyright action against Accolade, Sega alleged not that Accolade’s games were substantially similar to its own,\(^\text{110}\) but that Accolade’s “intermediate copies” of its games constituted an infringement.\(^\text{111}\) The issue was one of first impression in the Ninth Circuit.\(^\text{112}\) The court rejected Accolade’s argument that intermediate copying did not constitute copyright infringement,\(^\text{113}\) holding instead that Accolade’s reverse engineering of Sega’s games was a fair use under section 107.\(^\text{114}\)

Section 107 is the copyright statute’s fair use provision, which recognizes that some activities which infringe upon an author’s exclusive rights\(^\text{115}\) are nevertheless desirable. The classic example of fair use is a newspaper’s reproduction of excerpts from a copyrighted book for purposes of review. While technically a violation of the right of reproduction,\(^\text{116}\) the use of the book for this purpose is beyond reproach.\(^\text{117}\)

\(^{106}\) Sega, 977 F.2d at 1515. Actually, Sega also included an initialization code (akin to a password) in a newer version of its console, creating a second hurdle for Accolade’s engineers. The hurdle was not high: during the reverse engineering process, Accolade had spotted a section of code in Sega’s games that did not interact with the console. Accolade correctly anticipated Sega’s use of the code in a later console and made a note of it. Id. at 1515-16. This factual variation is immaterial to the decision.

\(^{107}\) The “printouts of resulting source code,” id. at 1515, are either copies of Sega’s programs, which are marketed only in object code form, or, more likely, derivative works. (Just as Virgil’s Aeneid may be translated from Latin into English—or French—in many ways, so may object code be translated into Pascal—or C—in many ways.) In either event, a copyright violation has occurred, either by violating Sega’s right of reproduction or adaptation. See id. at 1526 (“Disassembly of object code necessarily entails copying.”) and 17 U.S.C. § 106(1), (2) (1988).

\(^{108}\) Sega, 977 F.2d at 1518, 1525 (“The record makes clear that disassembly is wholesale copying.”).

\(^{109}\) Id. at 1515.

\(^{110}\) They probably were not. Any similarities beyond the nonprotectable interface specifications were at high levels of abstraction. Id. at 1516. Both companies, for example, marketed football video games with the names of famous National Football League figures in their titles. Id. at 1523 (Sega’s Joe Montana Football and Accolade’s Mike Ditka Power Football).

\(^{111}\) Id. at 1517-18. Sega also sued for “trademark infringement and false designation of origin . . . .” Id. at 1516. The copyright claim was actually included only in Sega’s amended complaint. Id.

\(^{112}\) Id. at 1519.

\(^{113}\) Id. at 1518-19. Accolade also argued that “disassembly” of object code was legal under both 17 U.S.C. § 102(b) (forbidding copyrights on ideas, processes, etc.) and under 17 U.S.C. § 117 (allowing computer users to copy programs into random access memory, which is necessary to run the software). The court rejected each of these arguments. Id. at 1519-20.

\(^{114}\) Id. at 1520.

\(^{115}\) In the United States, the author’s exclusive rights are those of reproduction, translation, distribution, performance, and display. 17 U.S.C. § 106 (1988).

\(^{116}\) 17 U.S.C. § 106(1).
Fair use is a defense to copyright infringement, and section 107 specifies the four factors to be weighed in a fair use analysis. These factors are:

1. the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;
2. the nature of the copyrighted work;
3. the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
4. the effect of the use upon the potential market for or value of the copyrighted work.\(^\text{118}\)

As to the amount of Sega's programs copied by Accolade, the court held, the third fair use factor clearly cuts in Sega's favor: Accolade copied the entire programs to discover their functional specifications.\(^\text{119}\) The remaining three factors, however, favored Accolade. On these, the court's analysis was driven by the special nature of computer programs, particularly object code. As the court itself noted, "the key to this case is that we are dealing with computer software, a relatively unexplored area in the world of copyright law."\(^\text{120}\)

1. The Purpose and Character of the Use

The first statutory factor of fair use, its purpose and character, runs in favor of those engaged in scholarship and research and against those who copy for commercial gain.\(^\text{121}\) The court found in favor of the copier Accolade, noting that although Accolade's "ultimate" purpose was commercial, its "direct" purpose in copying was "to study the functional requirements for [console] compatibility."\(^\text{122}\) Because the case revolved on whether intermediate copying was illegal, the distinction is necessary: Accolade's use of Sega's games occurs at a relatively early stage in program development. Conceptually, the development of a computer program's source code may be seen in two stages—one to write the purely expressive parts of the program, the other to ensure that the interface between the software and the hardware is a good

---

\(^{117}\) Like every rule, however, this one has its exception: In Harper & Row, Publishers, Inc. v. Nation Enters., 471 U.S. 599 (1985), Nation magazine obtained a copy of former President Ford's memoirs, in which Time Magazine had exclusive rights, and published the parts of it relating to the Nixon pardon first, thus scooping Time. The Supreme Court held that this use of the work, which the Nation had purloined, did not qualify as a fair use. Id.


\(^{119}\) Sega, 977 F.2d at 1526.

\(^{120}\) Id. at 1527. The court seems most troubled by the prospect of allowing ideas to remain legally sealed within unreadable object code, which "defeats the fundamental purpose of the Copyright Act—to encourage the production of original works by protecting the expressive elements of those works while leaving the ideas, facts, and functional concepts in the public domain for others to build on." Id.


\(^{122}\) Sega, 977 F.2d at 1522.
one. In the second stage, the programmer's task is to make the software operate according to the laws (specifications) of both the operating system and the microprocessor.

For conventional works of literature, the distinction does not make much sense because the "use" of the copier occurs at the level of output, not an intermediate level. In Harper & Row, Publishers, Inc. v. Nation Enterprises, for example, Nation magazine reprinted sections of Ford's memoirs in which Time magazine had exclusive rights. The Court concluded that the use was commercial and held for the plaintiff. In Sega, the court's identification of the precise nature of Accolade's use reflects the realities of software production. Perhaps more important is the court's reading of the fair use provisions, which presuppose desirable uses such as scholarship and research: "No other method of studying [the interface] requirements was available to Accolade" other than decompilation.

2. Nature of the Copyrighted Work

In general, works of fiction receive greater protection than factual works because the level of expression in them is higher. Another way of stating this is that unprotectable expression (e.g., facts and ideas) is less common in fictional works. The second statement has implications for both factual and functional works, including computer software.

The Sega court begins by stating its approval of the Computer Associates decision, noting that "many aspects of the program are not protected by copyright." Moreover, these unprotectable elements, which are plainly visible when the work is a book or a play, are hidden from the public when the work is a program marketed only in object code. Disassembly or decompilation is thus "necessary . . . to understand the functional requirements for [console] compatibility." The court held that the second fair use factor favored Accolade after facing

\[123\] As the Sega court notes, most cases that touch on the issue of intermediate copying hold only on the issue of whether defendant's final product infringes the plaintiff's copyright. Sega, 977 F.2d at 1518-19.


\[125\] Verbatim passages of the memoirs accounted for 13% of the article that appeared in the Nation. Id. at 548.

\[126\] Id. at 562.

\[127\] Id. at 569.

\[128\] Sega Enters. v. Accolade, 977 F.2d 1510, 1522 (9th Cir. 1992).


\[130\] Sega, 977 F.2d at 1525.

\[131\] Id. While it is true that some people are able to interpret object code, the court notes that even experienced programmers cannot interpret an entire program in object code due to its inordinate length. Even without a decompiler, then, copying the program is still necessary. Id.

\[132\] Id. at 1526.
this fact about object code: "Sega's video game programs contain unprotected aspects that cannot be examined without copying."\(^{133}\)

3. Effect on Potential Market

The standard formulation of the fourth fair use factor—the potential effect of the use on the market for plaintiff's work\(^{134}\)—is "if [the challenged use] should become widespread, it would adversely affect the potential market for the copyrighted work."\(^{135}\) Again the court distinguished the intermediate use from the ultimate purpose of Accolade's copying. Reverse engineering to learn Sega's interface specification, the court held, "simply enables the copier to enter the market for works of the same type as the copied work."\(^{136}\) The court states that this is not tantamount to usurping the market for the first work, as was the case in *Harper & Row*.\(^{137}\) In that case, the court notes, a person wanting to read presidential memoirs would buy only one copy of them, while video game users typically purchase many games. At bottom, however, the court remained troubled by the use of object code in light of the purposes of copyright. In its final observation on the fourth fair use factor, the court states that the "attempt to monopolize the market by making it impossible for others to compete runs counter to the statutory purpose of promoting creative expression . . . ."\(^{138}\)

V. EC Law

In the EC, the law governing the decompilation of computer software is the European Council Directive of May 14, 1991, on the Legal Protection of Computer Programs (Directive).\(^{139}\) Aimed at harmonizing EC copyright law with respect to computer programs\(^{140}\) and at the smooth functioning of the EC software industry,\(^{141}\) the Directive

\(^{133}\) Id.
\(^{135}\) *Sega*, 977 F.2d at 1523 (citing *Sony Corp. v. Universal City Studios*, 464 U.S. 417, 451 (1984)).
\(^{136}\) Id.
\(^{137}\) *Sega*, 977 F.2d at 1523-24.
\(^{138}\) *Id.*
\(^{140}\) Id.
\(^{142}\) See Directive, supra note 139, pmbl., para. 4 ("[C]ertain differences in the legal protection of computer programs offered by the laws of the Member States have direct and negative effect on the functioning of the common market . . . .").
\(^{143}\) See Directive, supra note 139, pmbl., para. 10 ("[P]hysical interconnection and interaction is required to permit all elements of software and hardware to work with other software and hardware and with users in all ways in which they are intended to function."); id. para. 3 ("[C]omputer programs are playing an increasingly important role in a broad range of industries and computer program technology can accordingly be considered as being of fundamental importance for the Community's industrial development.").
allows the rightful possessor\textsuperscript{142} of software to decompile it "to achieve interoperability."	extsuperscript{143} The right is constrained, however. First, there is no right to decompile where "the information necessary to achieve interoperability [is] readily available..."\textsuperscript{144} Second, decompilation may not "be used for goals other than to achieve the interoperability of the independently created computer program."\textsuperscript{145} Third, the right is limited to "obtain[ing] the information,"\textsuperscript{146} which may not then "be used for the development, production, or marketing of a computer program substantially similar in its expression" to the original (decompiled) program.\textsuperscript{147} This last proscription is somewhat troubling. The first part suggests that achieving interoperability is the \textit{only} purpose for which a program may be decompiled. If the limitation is complete, then it would follow that the use of the decompiled information, to develop \textit{any} program, except to achieve compatibility, would be forbidden. Instead, the second part of the proscription says that the information may not be used to develop a substantially similar program. The Directive does not define the phrase "substantially similar," which is pregnant with meaning in American copyright law.\textsuperscript{148} Most likely the second part of the proscription is overkill. Article 6.1 grants the right of decompilation to rightful possessors, while Article 6.2 outlines the limits of that right. Indeed, Article 6.2(a) circumscribes it entirely by stating that the use of decompiled information is unlawful "other than to achieve the interoperability of the independently created program."\textsuperscript{149} Interoperability, much like compatibility, is "defined as the ability to exchange information and mutually to use the information which has been exchanged" and refers to "functional interconnection and interaction,"\textsuperscript{150} which is necessary for "all elements of software and hardware to work with other software and hardware."\textsuperscript{151} The Directive specifies that those parts of a program which implement interoperability are "generally known as interfaces,"\textsuperscript{152} and limits decompiling to those sections of the program.\textsuperscript{153}

\textsuperscript{142} "[L]icensee or... another person having a right to use a copy of a program..."

\textsuperscript{143} Id. art. 6.1.b.

\textsuperscript{144} Id.

\textsuperscript{145} Id. art. 6.2.a.

\textsuperscript{146} Id. art. 6.1 (emphasis added).

\textsuperscript{147} Id. art. 6.2(c).

\textsuperscript{148} Where a plaintiff cannot show that the defendant copied directly, he may instead show that defendant had access to his work and that the defendant's work is substantially similar to the plaintiff's. See Feist Publications, Inc. v. Rural Tel. Serv. Co., Inc., 499 U.S. 340 (1991).

\textsuperscript{149} Directive, supra note 139, art. 6.2(a).

\textsuperscript{150} Id. pmbl., para 12.

\textsuperscript{151} Id. pmbl., para. 10.

\textsuperscript{152} Id. pmbl., para. 11.

\textsuperscript{153} Id. art. 6.1.c. The provisions appear ineffective. The reason programmers decompile software is that the object code is unintelligible streams of ones and zeros. As such, it offers no clues as to the location of the interface sections. That is, the code must first be decom-
Perhaps one of the most striking features of the Directive is Article 9.1, which voids any contractual provision contrary to the provisions in Article 6 (decompilation). Thus, the provisions in many American shrink-wrap licenses that flatly prohibit any form of reverse engineering would be void in the EC.\textsuperscript{154} The provision will strike American readers as odd because of the extent to which it goes beyond the preemption laws. In the United States, federal copyright law preempts state law only to the extent that the latter frustrates the former.\textsuperscript{155} But preemption is a far cry from nullifying agreements between private parties, who are free to modify, augment, or waive rights granted under the federal copyright scheme.\textsuperscript{156}

Aside from its decompilation and contractual provisions, the Directive is a close approximation to American copyright law.\textsuperscript{157} This should not be surprising. The EC "is fully committed to the promotion of international standardization,"\textsuperscript{158} and the United States has the most thorough and influential laws regarding the protection of computer programs.\textsuperscript{159} Countries turn to U.S. law in high technology al-

\textsuperscript{154} Decompilation, of course, is only one way to reverse engineer a program. Whether the provisions in the shrink-wrap licenses that forbid reverse engineering would be void \textit{in toto} or merely blue penciled is an issue beyond the scope of this Comment.


\textsuperscript{156} \textit{But see} Vault Corp. v. Quaid Software Ltd., 847 F.2d 255, 270 (5th Cir. 1988) (holding that federal copyright law preempts those provisions of Louisiana's Software License Act that allow software producers to forbid, via license, program adaptation via reverse engineering).


The Directive is American in a significant omission: it does not mention moral rights, despite the fact that they are protected in 10 of the 12 Member States. \textit{See} Christopher Voss, \textit{The Legal Protection of Computer Programs in the European Economic Community}, 11 \textit{COMPUTER/ L.J.} 441, 443 n.13 (1992).

\textsuperscript{158} Directive, \textit{supra} note 139, pmbl., para. 9.

\textsuperscript{159} "Governments in [foreign] countries look to the United States as a leader in this area because our law is better developed; we've been pursuing case law in the software protection area for much longer than most other nations." Symposium, \textit{Copyright Protection: Has Look and Feel Crashed?}, 11 \textit{CARDOZO ARTS & ENT. L.J.} 721, 748 (1993) (quoting Anthony Lawrence Clapes).
The biggest difference between U.S. and EC software law, however, is the treatment of fair use. The EC Directive, aside from the narrow setting of interoperability, has no fair use provision. ¹⁶¹ By contrast, U.S. copyright law contains a general fair use provision that is applied according to equitable principles. ¹⁶² Moreover, the latest copyright decision handed down by the U.S. Supreme Court broadens the scope of an important fair use factor ¹⁶³—the nature of the use. ¹⁶⁴ In *Campbell v. Acuff-Rose Music, Inc.*, ¹⁶⁵ the Court considered whether 2 Live Crew's version of Roy Orbison's "Pretty Woman" was protected by fair use. ¹⁶⁶ In construing the first factor of fair use—"the purpose and character of the use, including whether such use is of a commercial nature" ¹⁶⁷—the Court stated that the analysis should focus on "whether and to what extent the new work is 'transformative.' "¹⁶⁸ The Court found in favor of fair use, reversing the 6th Circuit's decision and criticizing its exclusive reliance on the commercial nature of the remake. ¹⁶⁹

The Supreme Court's endorsement of examining the transformative nature of a work, rather than its commercial nature, is clearly significant to reverse engineering. In *Sega*, defendant Accolade manufactured entirely new video games, surely a transformation (except for the initialization code) of Sega's original games. ¹⁷⁰ The point is not to outline the ramifications of *Campbell*, only to indicate that the scope of fair use in the United States is elastic (and possibly growing) in relation to that in the EC.

---

¹⁶⁰ A recent English decision on computer software infringement turns to both the Sega and Computer Assocs. cases for guidance on the nature and copyrightability of software. John Richardson Computers, Ltd. v. Flanders, [1993] F.S.R. 497 (Eng. C.A.). Though handed down two months after the EC Directive was to have gone in effect, the case does not even mention it.


¹⁶¹ See generally Directive, supra note 139.

¹⁶² See, e.g., Sega Enters. v. Accolade, 977 F.2d 1510 (9th Cir. 1992).


¹⁶⁵ 114 S. Ct. 1164 (1994).

¹⁶⁶ "It is uncontested here that 2 Live Crew's song would be considered an infringement... but for a finding of fair use through parody." *Campbell*, 114 S. Ct. at 1169.


¹⁶⁸ *Campbell*, 114 S. Ct. at 1171.

¹⁶⁹ *Id.* at 1173-74. "The mere fact that a use is educational and not for profit does not insulate it from a finding of infringement, any more than the commercial nature of a use bars a finding of fairness." *Id.* at 1174.

¹⁷⁰ As to the fourth factor of fair use—the effect on the potential market—the Supreme Court stated that 2 Live Crew's version of "Pretty Woman" would not affect the market for the original "because the parody and the original usually serve different market functions." *Id.* at 1178. This may not be the case with most video games.
VI. Conclusion

If Japan adopts the EC and U.S. position on reverse engineering for the purpose of achieving functional compatibility, it will follow through on its proposal despite U.S. resistance. Assuming Japan does this, it is unclear how much leeway it will give to reverse engineers of software. On one hand, U.S. law is more permissive than the EC Directive appears to be; on the other, Japanese copyright law has no fair use provision, so that to create one now would be to leap into vast and unchartered territory.

But reverse engineering in Japan, at least for purposes of achieving compatibility, is far less of a threat to the U.S. software industry than the potential reach of the abstraction-filtration-comparison test at home, which was designed as a guide for district courts in software infringement cases. Even with a broad right of reverse engineering, a second comer has no guarantee that his eventual product will pass a substantial similarity test. Had Accolade, for example, reverse engineered and analyzed Sega's games in toto and then marketed nearly identical games, a finding of infringement would have been nearly inevitable. The issue in Sega was merely whether Accolade's intermediate copies were lawful under fair use. Substantial similarity analyses apply to final products, which is a separate issue. Today, filtration analysis is performed just prior to a court's comparison of protectable elements of software. At the filtration stage, many venerable copyright doctrines are applied to whittle away expression that is not protectable by copyright or that is not dispositive of copying.

These doctrines should be used with great care. In particular, the scenes a faire doctrine, which disallows protection of shopworn conventions, lends itself to abuse. As Judge Learned Hand pointed out, a threadbare plot may be expressed in a variety of ways. When applying the scenes a faire doctrine, courts should keep in mind that cliches and conventions are often unoriginal only at the level of their ideas, not their particular expression. The presence of a cliched scene or section of code may not be dispositive of copying, but this is true only so far as the scene or section is considered in its entirety. The scene may permit a large number of possible expressions, so that excluding plaintiff's particular expression from the substantial similarity analysis is improper.

So, too, with a defendant's claims that efficiency is the supreme dictator of program writing. Courts should demand an explanation of

---


173 Sheldon v. Metro-Goldwyn Pictures Corp., 81 F.2d 49, 54 (2d Cir. 1936).
efficiency when it is proffered as a defense. Does efficiency mean the program's speed of execution? Or does it concern the way in which information is presented to the user? Or that for a given task, the source code version results in the least object code? My hunch is that the more precise the definition of efficiency, the further the line of copyrightability will lie from literal expression. That is, a vague notion of efficiency will tend to bar protection of a greater variety of code. On the other hand, as the definition of efficiency becomes more precise, it will eventually take on the look of a post hoc justification. Courts should thus ask the additional question of exactly how important is the goal of efficiency.

Courts should critically examine merger defenses as well. As a practical matter, the merger defense has succeeded where the total expression has been small and the purpose of the expression has been well defined. Today's typical program, by contrast, is planetary. The expanding body of languages, commands, functions, and procedures, which will continue to grow as microprocessors continue to advance, means that a task or idea may be implemented in an increasing number of ways. Thus, the defense of merger should be met with some skepticism.

Finally, courts should be careful not to conflate the analysis of protectable subject matter with that of what constitutes proper evidence of copying. There is naturally a lot of overlap, but the distinction should be kept in mind. As the inevitable Learned Hand noted, the fact that an expression has its provenance in public domain material only weakens the inference that copying occurred. It does not destroy the inference. The danger becomes apparent by imagining a database. Facts are not copyrightable, but compilations of facts are. Yet an aggressive application of the filtration analysis to a database could leave nothing for comparison analysis, thus, absolving a defendant where his liability may be clear. The totality of the work would be missed. Courts should be wary of attempts to nick and scratch a program out of its legal existence.

At bottom, object code is functional, so that the copyright holder, who obtained a monopoly only in its expression, should not complain when others wish to examine its underlying ideas. But U.S. courts should see to it that the expression gets the protection it deserves. The world, including Japan, will be watching.

JOHN ESPENSHADE TITUS

175 Sheldon v. Metro-Goldwyn Pictures Corp., 81 F.2d 49, 54 (2d Cir. 1936).