UIC Law Review

Volume 28 | Issue 1 Article 3

Fall 1994

Legitimizing Decompilation of Computer Software under Copyright Law: A Square Peg in Search of a Square Hole, 28 J. Marshall L. Rev. 105 (1994)

Allan M. Soobert

LEGITIMIZING DECOMPILATION OF COMPUTER SOFTWARE UNDER COPYRIGHT LAW: A SQUARE PEG IN SEARCH OF A SQUARE HOLE

ALLAN M. SOOBERT*

INTRODUCTION

Computer programs involve subject matter that has pushed the limits of copyright law for many years. Recently, the scope of these limits has been challenged by the process of reverse engineering.[1] One of the many analytical steps involved in reverse engineering is a process called decompilation.[2] Decompilation involves copying a computer program and subsequently translating the program from a language that only a machine can understand to a language that a human can read with ease. Since this process requires the copying of a copyrighted program, decompilation has created pressing issues under copyright law. Namely, the question has arisen as to whether this increasingly significant industry practice can be accommodated under copyright law.[3]

This Article provides an analysis of the issues raised by reverse engineering and the decompilation of computer programs. Section I presents an overview of computer programs and decomp-

---

* LL.M. Candidate, The George Washington University, National Law Center; J.D., Franklin Pierce Law Center, 1993; M.S.E.E. George Mason University, 1990; B.S.E.E., Pennsylvania State University, 1986. Mr. Soobert is an associate of the law firm of Dorsey & Whitney in Washington, D.C., and has several years of engineering experience in computer simulation and modeling of digital communication systems. The opinions expressed herein are solely those of the author.

1. The term "reverse engineering" is defined as the technique of studying a product to determine how it is made. *See* Kewanee Oil Co. v. Bicron Corp., 416 U.S. 470, 476 (1974) (stating that reverse engineering includes "starting with the known product and working backward to divine the process which aided in its development").

2. *See generally* OFFICE OF TECHNOLOGY ASSESSMENT, FINDING A BALANCE: COMPUTER SOFTWARE, INTELLECTUAL PROPERTY, AND THE CHALLENGE OF TECHNOLOGICAL CHANGE 7 (1992) [hereinafter OTA REPORT].

3. Computer Assocs. Int'l, Inc. v. Altai Inc., 23 U.S.P.Q.2d (BNA) 1241, 1257 (2d Cir. 1992). Copyright law is "not ideally suited to the highly dynamic technology of computer science. Thus far, many of the decisions in this area reflect the courts' attempt to fit the proverbial square peg in a round hole." *Id.* By proposing a statutory solution, this Article solves the problem of forcing the square peg of decompilation in the round hole of copyright law.

ilation, and their application to current copyright law. Section II illustrates the struggle that courts face in balancing the need to protect computer programs on the one hand, while accommodating reverse engineering and decompilation on the other. Section III critiques the courts' solution to the issues at hand, pointing out how the courts' approach has stretched an inadequate statutory framework too far. Finally, Section IV of this Article calls for the creation of a statutory provision which exempts decompilation from the scope of copyright infringement under limited circumstances. In particular, this Article proposes that the practice of decompilation be legitimized by placing it squarely within the provisions of copyright law.

The proposed decompilation exception is carefully tailored to permit decompilation without unduly diminishing the protection for computer programs. This exception is, therefore, protective enough to prevent piracy, and yet, permits developers and programmers to access a program's ideas and unprotected elements in order to develop new programs. By explicitly accommodating decompilation, the proposed exception allows the software industry and consumers alike to benefit from cumulative innovation, standardization, and the creation of compatible programs.

I. BACKGROUND

A. Overview of Computer Programs and Decompilation

1. Source Code v. Object Code

Computer programs consist of sets of instructions that direct the operation of a computer.[4] Such programs are created by writing these instructions line-by-line, which together creates "source code." The source code is typically written in a high-level computer programming language that can be read and understood by humans.[5] The source code must, in turn, be converted into ma-

---

4. *See* Steven R. Englund, *Idea, Process, or Protected Expression?: Determining the Scope of Copyright Protection of the Structure of Computer Programs*, 88 MICH. L. REV. 866, 908 (1990). See *also infra* note 19 for a definition of "computer program" under copyright law. Computer programs may generally be divided into two categories: (i) operating system programs and (ii) application programs. Operating system programs include such programs as DOS, XENIX and OS/2. Such programs direct the operation of a computer's hardware components in, for example, using memory and starting and stopping application programs. *See* Lotus Dev. Corp. v. Paperback Software Int'l, 15 U.S.P.Q.2d (BNA) 1577, 1579 (D. Mass. 1990). Application programs are programs that allow a computer user to perform specific tasks, such as engaging in word processing, database management, or spread sheet calculations or playing video games. *Id.*

5. *See* Christopher M. Mislow, *Computer Microcode: Testing the Limits of Software Copyrightability*, 65 B.U. L. REV. 733, 743-44 (1985). Most high-level languages, such as BASIC, COBOL, FORTRAN, Pascal and C, consist of instructions and

chine-readable form so that it can be understood and executed by a computer.[6] This conversion process is called "compilation" and involves translating or compiling the source code into machine language or "object code," which can be understood and executed by a computer.[7]

The object code that is executed by the computer is a binary code consisting of "ones" and "zeros" that represent, for example, the respective "on" and "off" states of switches in a computer chip. Different strings of ones and zeros form commands that instruct the computer to perform certain basic operations, which may simply involve adding or multiplying two numbers together. Even simple instructions and commands may, however, require numerous lines of object code to complete the operation. Thus, to the human eye, the object code for a complete program can be overwhelming and incomprehensible, with the lines of code appearing as a continuing series of ones and zeros strung together, page after page. Consequently, although computer programs can be written in machine language, programmers rarely do so since the process is so tedious and time-consuming.[8]

---

commands that crudely resemble the English language, and thus, can readily be understood by humans. *Id.* Other languages, such as assembly language, more closely resemble machine-readable code. *Id.*; *see also* Gary R. Ignatin, Comment, *Let the Hackers Hack: Allowing the Reverse Engineering of Copyrighted Computer Programs to Achieve Compatibility*, 140 U. PA. L. REV. 1999, 2002 n.8 (1992).

6. A number of layers of software exist in a given computer. The application programs typically run at the highest level of software operating within a computer. The operating system programs lie beneath such application programs, coordinating the operation of the application. The lowest level of software is the "microcode," which "takes machine language instructions and converts them to the series of physical signals necessary to control the circuits of the computer." Ignatin, *supra* note 5, at 2002. The microcode resides permanently in the computer and "consists of a series of instructions that tell a microprocessor which of its thousands of transistors to actuate in order to perform the tasks directed by the [application programs]." *Id.* at n.13 (quoting NEC v. Intel, 10 U.S.P.Q.2d (BNA) 1177, 1178 (N.D. Cal. 1989)). Thus, microcode is technically distinguishable from an operating system or application program's machine language. For purposes of this paper, however, this distinction makes little difference. Accordingly, machine language is used herein to refer to both types of code.

7. *See* Dennis S. Karjala, *Copyright, Computer Software, and the New Protectionism*, 28 JURIMETRICS J. 33, 37 (1987). Technically, the translation process can be accomplished by either a compiler or an interpreter program. An interpreter program is a simultaneous translator that works in conjunction with the application program every time the application program is run, carrying out the instructions of the program one step at a time. *Paperback*, 15 U.S.P.Q.2d (BNA) at 1580. A compiler, in contrast, converts or translates the source code once and for all into machine language, after which the translated program can be executed without the need for any further resort to the compiler. *Id.*

8. Karjala, *supra* note 7, at 37. Instead, programmers write programs using a high-level programming language to create source code.

2. Decompilation

Decompilation is the process used in the reverse engineering of computer programs, which retranslates or "decompiles" object code into source code.[9] This process allows a developer or programmer, who is otherwise unable to read and study the program's object code, to transform the program into more easily understood source code. Although the process may be used for a number of reasons, the process is typically used to make a copy of the original code and modify this copy to create a product that will compete commercially with the original.[10]

B. Overview of Copyright Law

1. Computer Programs as Copyrightable Subject Matter

Before the copyright laws were revised in 1976,[11] Congress created a National Commission on New Technological Uses of Copyrighted Works (CONTU) to explore the interrelationship between law and advancing technology and to propose amendments to the copyright law that would better protect computer programs.[12] After three years of researching and analyzing a number of issues, CONTU recommended that computer programs be protected under copyright law, with limited amendments to the Copyright Act of 1976.[13] In particular, CONTU recommended

---

9. OTA REPORT, *supra* note 2. The term "disassembly" is frequently used interchangeably with the term "decompilation." Decompilation is the process where a high-level language is derived from a machine-language program. Disassembly, in contrast, merely transforms machine language into assembly language and is considered somewhat simpler than decompilation because of the "one-to-one correspondence between machine language statements and assembly language statements." *Id.* Consequently, decompilation typically begins with disassembly and is followed by a process of matching patterns of assembly language to higher level constructs in order to create the original source code. *Id.* Notwithstanding this distinction, the terms are used herein interchangeably.

10. *See* Arthur R. Miller, *Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU?*, 106 HARV. L. REV. 977, 1014 (1993) (describing recreation of lost source code and "debugging" as uses that are legitimate under the current copyright law).

11. *See* 17 U.S.C. §§ 101-810 (1976).

12. NATIONAL COMMISSION ON NEW TECHNOLOGICAL USES OF COPYRIGHTED WORKS, Pub. L. No. 93-573, 88 Stat. 1873 (1974) [hereinafter CONTU]; *see also* H.R. REP. NO. 1476, 94th Cong., 2d Sess. 54, at 116 *reprinted in* 1876 U.S.C.C.A.N. 5659, 5731 [hereinafter H.R. REP. NO. 1476]; Miller, *supra* note 10; Peter S. Menell, *Tailoring Legal Protection for Computer Software*, 39 STAN. L. REV. 1329 (1987) [hereinafter Menell, *Tailoring Legal Protection*]; Paula Samuelson, *CONTU Revisited: The Case Against Copyright Protection for Computer Programs in Machine-Readable Form*, 1984 DUKE L. J. 663 [hereinafter Samuelson, *CONTU Revisited*].

13. *See generally* CONTU FINAL REPORT ON NEW TECHNOLOGICAL USES OF COPYRIGHTED WORKS (1979) [hereinafter CONTU FINAL REPORT].

inter alia that a definition of "computer program" be added to § 101 of the Copyright Act.[14] Based on CONTU's recommendations, Congress amended the Copyright Act in 1980 to unequivocally reconfirm that copyright protection extends to computer programs.[15] As a result, the Copyright Act clearly encompasses computer programs as copyrightable subject matter.

Under the current scheme of protection, the Copyright Act extends protection to "original works of authorship fixed in any tangible medium of expression."[16] The broad categories of works protected under the act include "literary works,"[17] which are defined as "works, other than audiovisual works, expressed in words, numbers, or verbal or numerical symbols or indicia, regardless of the nature of the material objects, such as books, periodicals, manuscripts, phonorecords, film tapes, disks, or cards, in which they are embodied."[18] Although computer programs are not specifically listed in this statutory definition, it is well established that computer programs are protected as literary works.[19] As such, the literal elements of a computer program, whether expressed in source code or object code form, are subject to copyright protection.[20] Thus, once a program is written and, for example, saved on a disk, the program is protected by copyright.[21]

---

14. CONTU FINAL REPORT, *supra* note 13, at 11, 14-15; *see also infra* note 15 (discussing specific amendments) and note 19 (defining "computer program").

15. Specifically, the amendments involved the addition of a definition of "computer program" in § 101, the deletion of the interim § 117, and the substitution of a new § 117 that now gives owners of copyrighted programs a limited right to make limited adaptations or modifications to a program and to make archival or back-up copies of a program. *See* Samuelson, *Creating a New Kind of Intellectual Property: Applying the Lessons of the Chip Law to Computer Programs*, 70 MINN. L. REV. 471, 474-75 n.12 (1985) [hereinafter Samuelson, *Sui Generis Protection*].

16. 17 U.S.C. § 102(a) (1990).

17. *Id.* In addition to literary works, the Copyright Act extends protection to a number of other works, including: musical works; dramatic works; pantomimes and choreographic works; pictorial, graphic and sculptural works; motion pictures and other audiovisual works; and sound recordings. *Id.*

18. 17 U.S.C. § 101 (1990).

19. H.R. REP. NO. 1476, *supra* note 11, *reprinted in* 1976 U.S.C.C.A.N. at 5667. A computer program, however, is itself defined under the Copyright Act as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result." 17 U.S.C. § 101 (1990).

20. *See Computer Assocs.*, 23 U.S.P.Q.2d (BNA) at 1249-51 (2d Cir. 1992); *see also* Whelan Assoc., Inc. v. Jaslow Dental Laboratory, Inc., 797 F.2d 1222, 1233 (3d Cir. 1986) (holding that copyright extends to the structure and logic of the program, even absent copying of the program's literal elements); Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1249 (3d Cir. 1983) (holding source code and object code copyrightable).

21. A number of exclusive rights are extended to an author of a work protected by copyright. In particular, the author has the exclusive right to reproduce and distribute copies of the copyrighted work, to prepare derivative works based upon the copyrighted work, and, in most cases, to display and perform the work publicly.

2. Idea v. Expression

Computer programs, like all other works of authorship, are not necessarily entitled to an unlimited scope of copyright protection. Indeed, it is a fundamental principle of copyright law that a copyright in a work does not protect the ideas embodied in that work and, instead, protects only the expression of such ideas.[22] This principle has been codified in § 102(b) of the Copyright Act, which provides: "[i]n no case does copyright 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 a work."[23] Others may, therefore, copy and use the ideas in a copyrighted work, provided the author's expression is not likewise appropriated.[24]

Finding the line that separates idea and expression is, however, not a simple task.[25] This task is even more difficult where the copyrighted work is a computer program. In fact, due to their essentially utilitarian nature, computer programs, when compared to more traditional aesthetic works, "hover even more closely to the elusive boundary line described in §102 (b)."[26] To help define this line, Judge Learned Hand proposed an abstraction test to separate ideas from expression in written or dramatic works that is often applied to computer programs:

See 17 U.S.C. § 106 (1990) (listing the exclusive rights in copyrighted works). Such rights exist as soon as the work is fixed in a tangible medium of expression and generally extend thereafter for fifty years after the death of the author. *See* 17 U.S.C. 302(a) (1990) (specifying the duration of copyright protection).

22. This principle is known as the idea-expression dichotomy. *See* Mazer v. Stein, 347 U.S. 201, 217 (1954); Baker v. Selden, 101 U.S. 99 (1880). Congress has made clear that this dichotomy applies with equal force to computer programs: "[s]ection 102(b) is intended . . . to make clear that the expression adopted by the programmer is the copyrightable element in computer program, and that the actual processes or methods embodied in the program are not within the scope of copyright law." *See* H.R. REP. NO. 1476, *supra* note 11, *reprinted in* 1976 U.S.C.-C.A.N. at 5670.

23. 17 U.S.C. § 102(b) (1990).

24. Where, however, an idea can be expressed in only a limited number of ways, the expression is said to have "merged" with the idea. In such instances, the expression is not subject to copyright protection, and thus, may be readily copied. *See* Baker v. Selden, 101 U.S. 99, 104 (1880) (holding that copyright protection does not extend to those aspects of a work that "must necessarily be used as incident to" the idea, system or process described in the work).

25. As Judge Learned Hand stated, "Nobody has ever been able to fix that boundary, and nobody ever can." Nichols v. Universal Pictures Co., 45 F.2d. 119, 121 (2d Cir. 1930), *cert. denied*, 282 U.S. 902 (1931); *see also Computer Assocs.*, 23 U.S.P.Q.2d (BNA) at 1250 (quoting Judge Learned Hand's remarks in *Nichols* and noting that "his convictions have remained firm").

26. *Computer Assocs.*, 23 U.S.P.Q.2d (BNA) at 1251.

> Upon any work . . . a great number of patterns of increasing generality will fit equally well, as more and more of the incident is left out. . . . [T]here is a point in this series of abstractions where they are no longer protected, since otherwise the playwright could prevent the use of his "ideas," to which, apart from their expressions, his property is never extended.[27]

Application of Judge Hand's abstraction test allows unprotectable ideas to be differentiated from protectable expression, and recognizes that a computer program may contain many distinct ideas.[28] Thus, by separating the program into manageable components, the abstraction test enables the boundaries of protectable expression to be discerned.[29]

Upon separating the computer program into manageable components, the unprotectable components of the program must be filtered from the protectable expression.[30] This step requires that the unprotectable ideas be "filtered" from: (i) expression necessarily incident to the idea, (ii) expression already in the public domain, (iii) expression dictated by external factors,[31] and (iv) expression not original to the programmer or author.[32] Using such an analysis, unprotected ideas and protectable expression can be separated from one another.[33]

---

27. *Nichols*, 45 F.2d at 121, *cert. denied*, 282 U.S. 902 (1931). The Second Circuit Court of Appeals has recently applied this test to computer programs. *See Computer Assocs.*, 23 U.S.P.Q.2d (BNA) at 1251.

28. *See* Atari Games Corp. v. Nintendo of Am. Inc., 24 U.S.P.Q.2d (BNA) 1015, 1020 (Fed. Cir. 1992) (citing *Computer Assocs.*, 23 U.S.P.Q.2d (BNA) at 1253).

29. 24 U.S.P.Q.2d (BNA) at 1020.

30. *Id.*

31. Such external factors include the computer's hardware specifications, compatibility with other programs, and demands of the computer program's industry. *Id.* (citing Plains Cotton Coop. Ass'n v. Goodpasture Computer Servs., 807 F.2d 1256 (5th Cir.), *cert. denied*, 484 U.S. 821 (1987); Harper & Row v. Nation Enters., 471 U.S. 539, 548 (1985); *Computer Assocs.*, 23 U.S.P.Q.2d (BNA) at 1253).

32. *Id.*

33. Although this approach may seem somewhat complex, simpler approaches have been criticized as overbroad and inaccurate. *See, e.g.*, Englund, *supra* note 4, at 881. For instance, in Whelan Assocs. v. Jaslow Dental Lab., Inc., 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987), the Court of Appeals for the Third Circuit proposed an approach that attempts to draw the line between idea and expression by defining the idea and expression by purpose. Such an approach, however, has been criticized as "descriptively inadequate" and as based on a "somewhat outdated appreciation of computer science," which fails to recognize that a program typically has many distinct ideas. *Computer Assocs.*, 23 U.S.P.Q.2d (BNA) at 1252; *see also*, Englund, *supra* note 4, at 881; Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 STAN. L. REV. 1045, 1052 (1989).

3. Fair Use

An author's exclusive ownership rights in a copyrighted work are qualified, in part, by the fair use provision in § 107 of the Copyright Act.[34] Section 107 establishes a defense to an otherwise valid claim of copyright infringement, listing four factors that must be considered in determining whether a particular use of a copyrighted work is fair.[35] In particular, § 107 lists the following factors that must be weighed in any fair use analysis:

> (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 substantially 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.[36]

The first statutory factor requires weighing the purpose and character of the use, focusing on whether such use is for a commercial or noncommercial purpose. The Supreme Court held that "every commercial use of copyrighted material is presumptively an unfair exploitation . . . of the copyright,"[37] and thus, "weighs against a finding of fair use."[38] Consequently, where a computer program is copied in order to make a competing commercial product, the use of the program may not be considered fair.[39]

The second statutory factor requires consideration of the nature of the copyrighted work. Such consideration has tradition-

---

34. *See* 17 U.S.C § 107 (1990).

35. The preamble of the fair use provision indicates the types of purposes for which reproduction of a copyrighted work may be considered fair, including such purposes as "criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research." *Id.*

36. *Id.*; *see also* WILLIAM F. PATRY, THE FAIR USE OF PRIVILEGE IN COPYRIGHT LAW (1985) (discussing the fair use provision and its underlying statutory factors).

37. Sony Corp. of Am. v. Universal Studios, 464 U.S. 417, 451 (1984). The distinction between commercial and noncommercial use is "not whether the sole motive of the use is monetary gain but whether the user stands to profit from exploitation of the copyrighted material without paying the customary price." *Harper & Row*, 471 U.S. at 562.

38. *See Harper & Row*, 471 U.S. at 562. Although a commercial purpose raises a presumption of unfairness, such a presumption may, in some cases, be rebutted by the characteristics of a particular commercial use. *See also* Hustler Magazine, Inc. v. Moral Majority, Inc., 230 U.S.P.Q. (BNA) 646, 651-52 (9th Cir. 1986) (quoting MCA, Inc. v. Wilson, 677 F.2d 180, 182 (2d Cir. 1980)) (holding that where the challenged use serves the public interest, the presumption of unfairness may be rebutted).

39. *But see* Sega Enters. Ltd. v. Accolade, Inc., 24 U.S.P.Q.2d (BNA) 1561 (9th Cir. 1992).

ally involved, for example, determining whether the copyrighted work has been published.[40] Unpublished works are afforded protection greater than those works which have been published; and, consequently, the scope of fair use is "narrower with respect to [such] unpublished works."[41] Thus, because computer programs in object code form may arguably be considered unpublished, such programs are likely subject to a narrower application of fair use.[42]

The third statutory factor requires examination of the amount and substantiality of the portion used in relation to the copyrighted work as a whole. Where the use involves copying the "heart" of a work, such use weighs against a finding of fair use.[43] Thus, in cases where the entire work is copied, the third statutory factor tends to weigh against a finding a fair use.[44] Therefore, when a computer program is copied, for example, during decompilation, a finding of fair use may not be warranted under traditional analysis.[45]

The fourth statutory factor is typically considered the most important of the four statutory factors, although it bears a close relationship to the first factor's inquiry into the purpose and character of the use.[46] The fourth factor requires consideration of the effect of the use on: (i) the potential market for the copyrighted work or (ii) the value of the copyrighted work. Because of this factor's importance, a use that supplants a copyrighted work may prove dispositive in a fair use analysis.[47] This fourth statutory factor, therefore, requires inquiry into whether the use, "should [it] become widespread, . . . would adversely affect the potential market for the copyrighted work."[48] Such adverse effects include diminishing potential sales, interfering with marketability, and usurping the market.[49] Thus, as under the first factor, copying in order to make a competing commercial product tends to weigh against a fair use finding.[50]

---

40. *See Harper*, 471 U.S. at 555-60.

41. *Id.* This greater protection arises from "the author's right to control the first public appearance of his expression." *Id.* at 564.

42. *See* Ignatin, *supra* note 5, at 2040 n.150 (arguing that software is unpublished even after it has been disseminated to the public).

43. *See Harper*, 471 U.S. at 564-66.

44. *Id; cf. Sony*, 464 U.S. at 447-55 (holding that home video recording for time-shifting purposes is fair use, notwithstanding that the entire copyrighted work is recorded or copied).

45. *See* Ignatin, *supra* note 5, at 2040 (arguing that reverse engineering fails *all four* fair use factors, including the third factor since the entire work is copied).

46. *See Harper*, 471 U.S. at 567.

47. *Id.*

48. *Sony*, 464 U.S. at 451.

49. *Hustler*, 230 U.S.P.Q. (BNA) at 654.

50. *But see Sega*, 24 U.S.P.Q.2d (BNA) at 1561.

4. § 117

As part of its recommendations to Congress, CONTU recommended the enactment of § 117 of the Copyright Act to ensure that owners of computer programs could use and place copies of their programs into their computer's memory without being subjected to potential liability for copyright infringement.[51] In general, § 117 provides a narrow exception for copying a computer program when such copying is essential to the legitimate use of the original program.[52] Due to its limited scope, § 117 typically does not protect someone who, for example, makes copies of a program for reverse engineering purposes.[53]

C. The Problem of Applying Copyright Law to Decompilation: Trying to Force a Square Peg Into a Round Hole

The current provisions of the Copyright Act do not explicitly accommodate the practice of decompilation. The courts have, therefore, struggled to find a way to justify the practice. In so doing, they attempt to stretch an inadequate statutory framework

---

51. *See* CONTU FINAL REPORT, *supra* note 13, at 16-18. Congress enacted section 117 essentially in the form proposed by CONTU. The enacted section extended the limited rights granted under section 117 to "owners" of copies of computer programs, rather than to "rightful possessors" of such copies, as proposed. *See* Richard H. Stern, *Section 117 of the Copyright Act: Charter of the Software Users' or an Illusory Promise?*, 7 W. NEW ENG. L. REV. 459, 460 (1985) (citing CONTU FINAL REPORT, *supra* note 12, at 12-13).

52. As enacted, section 117 provides:

> Notwithstanding the provisions of section 106 [which lists the exclusive rights of a copyright owner], it is not an infringement for the owner of a copy of a computer program to make or authorize the making of another copy or adaptation of that computer program provided:
>
> (1) that such a new copy or adaptation is created as an essential step in the utilization of the computer program in conjunction with a machine and that it is used in no other manner, or
>
> (2) that such new copy or adaptation is for archival purposes only and that all archival copies are destroyed in the event that continued possession of the computer program should cease to be rightful.

17 U.S.C. § 117 (1990).

53. *See Sega*, 24 U.S.P.Q.2d (BNA) at 1568; *see also* Miller, *supra* note 10, at 1016 n.185 (citing Allen-Myland, Inc. v. Int'l Business Machs. Corp., 746 F. Supp. 520, 535-36 (E.D. Pa. 1990) (finding the use of a "library" of copied microcode not to be an "essential step," and "adaptation," or "archival"); Micro-Sparc, Inc. v. Amtype Corp., 592 F. Supp. 33, 35 (D. Mass. 1984) (finding that the "essential step" provision in section 117 is limited to the rightful possessor, not to a third party); Atari v. JS&A Group, Inc., 597 F. Supp. 5, 9-10 (N.D. Ill. 1983) (finding that the "archival" provision in section 117 does not apply to programs that cannot be damaged or erased)); *cf.* Vault Corp. v. Quaid Software Ltd., 7 U.S.P.Q.2d (BNA) 1281 (5th Cir. 1988) (holding that section 117 preempted a Louisiana statute that authorized license agreements to prohibit decompilation of computer programs).

too far. Consequently, a different solution is needed. The following sections analyze the courts' struggle in handling this problem, illustrate the flaws in their approach and, accordingly, call for a different solution that accommodates decompilation by statute.

II. THE EMERGING STRUGGLE: WEIGHING THE NEED FOR DECOMPILATION AGAINST THE NEED TO PROTECT COMPUTER SOFTWARE

A. One Side of the Scale: The Need for Decompilation

1. Access to Ideas

Since copyright law protects an author's expression of ideas rather than the ideas themselves, others are generally free to extract and use the ideas from a given work, provided that the expression of those ideas is likewise not copied.[54] For example, a person interested in writing a legal novel may study the works of Scott Turow and John Grisham to extract the novelists' ideas in creating a new legal thriller. Likewise, someone interested in creating a movie about international espionage can look to books written by Tom Clancy for suggestions.[55] Through study and analysis, therefore, anyone is free to extract and use the ideas that are expressed in these works simply by reading the works.

For computer programs, however, such study and analysis cannot, without more, be accomplished. Computer programs are typically sold and disseminated to the public in object code form. Thus, unlike most traditional literary works, such computer programs cannot be read and understood by humans without laborious effort.[56] Instead, object code must be decompiled in order to be retranslated into a form understandable by humans.[57] Decompilation is, therefore, vital to identifying the ideas and functional elements embodied in a computer program.

2. Compatibility

Closely related to the need to access the ideas in a computer program is the need to create compatible programs. In general, a compatible program is a computer program that is capable of working with another program. Compatibility is an important part of the software industry since many programs are specifically developed to interoperate with other programs.[58] For example,

---

54. *See* 17 U.S.C. § 102(b) (1990); *see also* Ignatin, *supra* note 5, at 2009-14.

55. Ignatin, *supra* note 5, at 2009-14.

56. *See supra* notes 4-8 and accompanying text (distinguishing source and object code).

57. *See supra* notes 9-10 and accompanying text (explaining decompilation).

58. *See* Ignatin, *supra* note 5, at 2025. For purposes of this Article,

assume a software developer seeks to write an application program that creates charts and plots. Such a program may prove particularly useful in plotting the results produced from other application programs, such as a program that performs spreadsheet calculations. Although the plotting program could run independently of the spreadsheet program, the two programs would be more useful if each could operate with the other.[59] Assuming interoperability is achieved, the output of the spreadsheet program could be used directly with the plotting program to produce graphical representations of the spreadsheet program's output.

One method of achieving such program compatibility would be to decompile the spreadsheet program to learn, for instance, the program's input and output requirements, its data formatting techniques, and the types of variables used in the program. Such information could then be incorporated into the plotting program, as it is created, so that the two programs would interoperate with one another. Under these circumstances, programmers would benefit from more choices of programs in which compatibility could be achieved. Software developers would likewise have more programs that would interoperate with one another, which would likely increase user demand for software with increased interoperational capabilities. In addition, program users would be presented with more programs to choose from and lower prices from the increased competition among software developers.[60]

As compatibility increases, the benefits of standardization and "network externalities" will also increase.[61] For instance, input and output formats will become standardized as more compatible programs are developed, allowing users to more freely transfer data between programs.[62] Likewise, users will benefit by

---

interoperability is synonymous with compatibility.

59. Ignatin, *supra* note 5, at 2025.

60. *Id.*

61. Network externalities are the benefits that accrue to computer users from the creation of large computer networks that make use of compatible and standardized application programs. *See* David Victor, *An Analysis of an Affirmative Defense for Reverse Engineering Within A System of Legal Protection for Computer Software*, 66 S. CAL. L. REV. 1705, 1726 (1993) (explaining that Sun Microsystems maintains liberal licensing policies, which encourage software developers to write programs for Sun Microsystems' networks).

62. *See* Joseph Farrell, *Standardization and Intellectual Property, Last Frontier Conference on Copyright Protection of Computer Software*, 30 JURIMETRICS J. 35, 36-39 (1989). Note, however, that standardization is not necessarily always socially or economically beneficial. For example, as the public becomes more accustomed to a specific standard, the more difficult it becomes to change to other more beneficial standards. The "QWERTY" keyboard layout is a classic example of such a case: although other layouts are available, adoption of another alternative is virtually impossible since the general public is not likely to be persuaded to change standards. *See* Miller, *supra* note 10, at 1030.

having access to a larger network as well as increased access to other software.[63] Such benefits, however, cannot be achieved without decompilation. Programmers and developers must be able to study and use, for example, common program interface requirements in order to reap these benefits. As a result, decompilation is an important tool for achieving compatibility, and thus, for benefitting from standardization and network externalities.

3. *Encouraging Cumulative Innovation*

In most areas of technology, new advancements are achieved by building on developments previously made and incorporated in existing technology.[64] That is, by making use of existing technology, new products and technologies may be developed by simply improving on those already in existence, without the need to "reinvent the wheel."[65] Such improvements in existing technology result in a process of sequential development that increases consumer satisfaction and accelerates the introduction of new products into an existing industry.[66] This process also allows producers and developers to make use of the "incremental value" that an initial innovation contributes to subsequent derivative products.[67] Making use of initial innovation and such incremental value is commonly referred to as "cumulative innovation."[68]

Such cumulative innovation is critical in the development of computer programs. Indeed, most members of the computer industry believe that the development of computer programs has

---

63. *See* Farrell, *supra* note 61, at 36-39. Such a benefit will also likely increase the standardization of human interface designs. These designs are often based on human factors engineering, which assesses the aesthetic, psychological, graphical and ergonometric factors with the greatest appeal to humans. This appeal is considered as the key driving force in the overall success of an audio-visual interface. *See* Victor, *supra* note 61, at 1727 n.115 (quoting Menell, *An Analysis*, *supra* note 33, at 1053-54 n.37).

64. *See* Victor, *supra* note 61, at 1722; *see also* Robert P. Merges & Richard R. Nelson, *On the Complex Economics of Patent Scope*, 90 COLUM. L. REV. 839, 880-81 (1990) (discussing such advancements in a broad range of industries, including aircraft, automobile, computer, electric lighting, and semiconductor industries).

65. *See, e.g.*, Ignatin, *supra* note 5, at 2030.

66. *See* Victor, *supra* note 61, at 1723.

67. *Id.*; *see also* Merges & Nelson, *supra* note 64, at 880-81. The incremental value in initial innovation can economically be attributed to: (i) the entire value of the derivative product in cases where the product could not have been developed without the initial innovation; (ii) the cost savings realized in cases where the initial innovation has reduced the cost of developing the derivative product; or (iii) the value of the derivative product's accelerated release into the marketplace. *See* Victor, *supra* note 61, at 1723.

68. *See* Victor, *supra* note 61, at 1722. Cumulative innovation is distinguished from "discrete innovation," which refers generally to advancements in technology that are independent of previous innovations. *Id.*

thrived on cumulative innovation in the past and should continue to do so.[69] In particular, these members maintain that the current software industry has resulted from an "evolutionary development" that has spurred "considerable amount[s] of software innovation."[70] However, in order for the software industry to keep pace with rapid changes in innovation, software developers must be able to build upon those developments already included in existing computer programs. Decompilation is the key step in this building process. As a result, software developers must be able to decompile existing computer programs in order to learn from existing technology and to continue promoting the progress of science and the useful arts.[71]

B. The Other Side of the Scale: The Importance of Protecting Computer Programs

1. Preventing Piracy

The arguments in favor of maintaining strong copyright protection of computer programs are generally based on a single theme: preventing piracy.[72] Any system of copyright protection for computer programs, that would, for example, permit limited instances of copying in order to access unprotected ideas, runs the risk of leaving programs with inadequate protection. Because this risk increases the likelihood of piracy, the system of protection must be carefully tailored to serve important goals such as maintaining the incentive to *independently* produce an innovative or creative expression of one's own.

Piracy is a real problem with computer programs since computer programs, by their nature, are easily duplicated. A pirate who seeks to duplicate a program can do so, in most cases, in minutes, if not seconds. The duplication may involve nothing more than copying the program from one disk to another. Thus, the pirate can "create" a copy of a program without incurring any of the development costs that were involved in creating the original program.[73] The pirate, therefore, can market a competing version of the original program almost immediately, by simply purchasing

---

69. *Id.* (citing Paula Samuelson and Robert J. Glushko, *Survey on the Look and Feel of Lawsuits*, 33 COMM. ASS'N COMPUTING MACHINERY 483, 483-85 (1990)).

70. Victor, *supra* note 61, at 1722.

71. Congress is mandated to fulfill this goal. *See* U.S. CONST. art. I, § 8, cl. 8 ("Congress shall have the Power . . . [t]o promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries. . . .").

72. *See e.g.*, Karjala, *supra* note 7, at 40; Miller, *supra* note 10, at 1026-29; Ignatin, *supra* note 5, at 2034-39.

73. Remarkably, such copying can be accomplished without even knowing anything about the program's code. *See* Karjala, *supra* note 7, at 40.

a single copy of the original program and generating duplicates. Having foregone the development costs involved in independent creation, the pirate can likewise charge a much lower price for its product.[74] In the end, the original creator's lead time in market entry is reduced to almost nothing. Consequently, since piracy is the source of such harmful effects, copyright protection of computer programs must involve means to prevent piracy.

2. Encouraging Initial Innovation

A corollary to maintaining a system of copyright protection that prevents piracy is the need for a system that provides incentives for initial innovation. Initial innovation can be encouraged by crafting a system of copyright protection that does not tolerate piracy.[75] This system of protection should condone innovation and, at the same time, condemn slavish copying. Such a system should seek to promote new developments in technology by accommodating developers' desires to maintain enough lead time to recoup their development costs.[76] Any system of copyright protection for computer programs should strive to meet these goals, regardless of whether the system also provides limited exemptions for archival, fair use, and decompilation.

For example, the computer industry benefits from cumulative innovation, because most programs and computer-related products build on previous developments in technology.[77] Many members of the computer industry believe that the industry thrives on cumulative innovation and, consequently, is harmed by an expanded scope of protection that prevents the creation of derivative computer programs.[78] The harm that results generally involves a slowed pace of computer program development. In turn, this decrease in development could create "performance gaps" between the hardware and software industries, with hardware products developing more rapidly than their software counterparts.[79]

Under the current system of protection, computer programs are entitled to a broad scope of copyright protection, and thus, others are, in most cases, restricted from decompiling a computer program to develop a *derivative* program.[80] Such protection gen-

---

74. *See id.*

75. *See, e.g.*, Karjala, *supra* note 7, at 40; Victor, *supra* note 61, at 1734-35.

76. *See* Karjala, *supra* note 7, at 58; *see also* Ignatin, *supra* note 5, at 2035.

77. *See* Merges and Nelson, *supra* note 64, at 880-81.

78. *See* Victor, *supra* note 61, at 1725.

79. *Id.* at 1724-25.

80. The right to prepare derivative works under copyright law is an exclusive right of the person who creates the original computer program. *See* 17 U.S.C. § 106(2) (1990); *cf.* NEC Corp. v. Intel Corp., 10 U.S.P.Q. (BNA) 1177 (N.D. Cal. 1989) (holding that NEC's final derivative program, which was created by

erally prevents others from copying the initial innovations of one program and using these innovations to create another program. Although two recent cases indicate that the Ninth Circuit is willing to permit limited decompilation of a program to study and use its *ideas* to achieve program compatibility, these cases do not permit, for example, the use of portions of a decompiled program to create an improved derivative program.[81] Consequently, the current copyright law, as it stands now, strongly encourages initial innovation and retards cumulative innovation by overprotecting computer programs. A more appropriate system of copyright protection for computer programs should strive to maintain the incentives for *both* initial innovation and cumulative innovation.

C. *The Need to Strike a Balance: The Benefits of Decompilation Must be Balanced with Maintaining the Protection of Computer Programs*

The system of copyright protection for computer programs must be carefully tailored so that it does not overly protect computer programs. The system should be protective enough to prevent piracy and, at the same time, permit developers and programmers to access ideas and other unprotected elements of a program. The system should likewise allow the software industry to benefit from standardization and the creation of compatible programs. Moreover, the system should not be overly protective of computer program so as to produce harmful effects, such as retarding cumulative innovation.

IV. THE COURTS' SEARCH FOR A SOLUTION: STRETCHING AN ADEQUATE STATUTORY FRAMEWORK TOO FAR?

A. *The Courts' Initial Reactions to Decompilation and Other Means of Copying*

1. *An Initial Reluctancy*

In a number of cases initially addressing issues of infringement of copyrighted computer programs, courts were repeatedly reluctant to accept the practice of studying and analyzing one computer program in order to develop another. For example, in *Midway Manufacturing Co. v. Strohon*,[82] the defendant, Slayton,

---

decompiling Intel's microcode, was not an infringement, even though the legality of the decompiled code was not decided); *see also infra* notes 107, 115 and accompanying text (explaining that NEC's initial decompilation of Intel's microcode may have been authorized under a license).

81. *See infra* notes 119-47, 148-54 and accompanying text (analyzing the *Atari* and *Sega* decisions and their reliance on fair use).

82. 564 F. Supp. 741 (N.D. Ill. 1983). Although video games are treated under

created a modified version of plaintiff's copyrighted PAC-MAN video game by *inter alia* increasing the game's playing speed so that the game would present a greater challenge to skilled players.[83] In addition to increasing the speed of play, the defendant also modified the appearance of the game's screen display by changing the shapes of the game's characters.[84] The *appearance* of the modified game was sufficiently different from that of PAC-MAN so that there was no copyright infringement of the *audiovisual* aspects of the game.[85] The computer program itself, however, in Slayton's modified game was substantially similar to the original PAC-MAN program.[86] This similarity resulted from the fact that Slayton had produced the modified code by simply "patching" lines of new code to the original code and subsequently encoding the combined code into memory chips.[87] Consequently, the district court, relying on traditional copyright principles, held that the modified code constituted copyright infringement even though Slayton had, in effect, improved the game.[88]

Similarly, in *Hubco Data Products Corp. v. Management Assistance, Inc.*,[89] a district court rejected the legality of reverse engineering of computer programs under copyright law. In *Hubco*, the copyright owner (MAI) held a copyright on an operating system, which was being marketed to its customers at different prices and with different capabilities.[90] The difference in price was justified by placing "governors" on less expensive versions of the operating system, which restricted memory capacity and the use of certain peripheral devices.[91] The alleged infringer (Hubco) offered MAI's customers a service in which the governors would be removed from the less expensive versions, thus enhancing the program's capabilities.[92] Initially, Hubco accomplished the conversion by reverse engineering the operating system through: (1)

---

copyright law as audiovisual works, rather than literary works, the case is both instructive and analogous to issues raised by similar uses of computer programs. *See* Karjala, *supra* note 7, at 49 n.54, 60.

83. *See Midway*, 564 F. Supp. at 744.

84. *Id.*

85. *See id.* at 748-49 (emphasis added); *see also* Stern, *Section 117*, *supra* note 51, at 474.

86. The court determined that eighty-nine percent of the original program was copied into the defendant's modified program. *See Midway*, 564 F. Supp. at 752.

87. *Id.; see also* Stern, *Section 117*, *supra* note 51, at 474 n.96 (explaining that "[a] 'patch' is a slight modification of a program, done without modifying or recompiling the rest of the program") (citation omitted).

88. *See* Karjala, *supra* note 7, at 61 (suggesting that, by creating a more challenging game for skilled players, the defendant supplied a social benefit).

89. 219 U.S.P.Q. (BNA) 450 (D. Idaho 1983).

90. *Id.*

91. *Id.*

92. *Id.*

making a printout of the operating system program; (2) "decoding" the program into code that could be more easily understood by humans; (3) performing a line-by-line comparison to locate the governors by inspection; and (4) modifying or deleting the governors that were found.[93] Later, Hubco supplied the customers with a computer program that essentially performed the same steps, with the copying and comparison of MAI's code occurring entirely within the computer.[94] The court held that both methods constituted copyright infringement, as both methods produced unauthorized copies of MAI's ungoverned code for comparison purposes.[95]

In *SAS Institute, Inc. v. S&H Computer Systems, Inc.*,[96] another district court similarly refused to accept reverse engineering as a legitimate practice. In *SAS Institute*, the defendant (S&H) converted plaintiff's (SAS) copyrighted statistical analysis program into another computer language so that the program could run on an incompatible computer.[97] Although S&H had obtained a license from SAS to only use the SAS program on an IBM computer, S&H translated the program's IBM compatible code into source code and then "retranslated" this source code into a VAX-compatible coding scheme.[98] Subsequently, printouts of the program in the VAX-coding scheme were output for study and analysis by S&H employees.[99] Thereafter, the S&H employees produced their own version of the code, which was held to be substantially similar to the original SAS program.[100] The court held that a number of copyright violations had occurred, including violations of SAS's exclusive right to make copies (e.g., the printouts) and to prepare derivative works (e.g., the S&H version of the code).[101]

The *Midco, Hubco,* and *SAS* cases illustrate the courts' initial reactions to decompilation. In particular, these cases show the courts' initial reluctance to embrace the practice.

---

93. *Id.* at 452.

94. *Hubco*, 219 U.S.P.Q. (BNA) at 452.

95. *Id.* at 456. The first method involved making a physical printout of MAI's program on paper, while the second method involved the computer reproduction of MAI's ungoverned program, which was compared against the governed system. *Id.*

96. 225 U.S.P.Q. (BNA) 916 (M. D. Tenn. 1985).

97. *Id.* at 919. Although S&H had obtained a license from SAS to only use the SAS program on an IBM computer, S&H translated the program's IBM compatible code into source code and then coded and re-translated this source code into a VAX-compatible coding scheme. Subsequently, printouts of the program in the VAX-coding scheme were output for study and analysis by S&H employees. Thereafter, the S&H employees produced their "own" S&H version of the code, which was substantially similar to the SAS program. *Id.* at 919-20.

98. *Id.*

99. *Id.*

100. *Id.* at 926-27.

101. *SAS Inst.*, 225 U.S.P.Q. (BNA) at 926-27.

2. A Trend Towards Encouraging the Practice

Although many courts were initially reluctant to accept decompilation of copyrighted computer programs as a legitimate practice, several other cases illustrate a trend towards encouraging the practice. For example, in *E.F. Johnson v. Uniden*,[102] the district court determined that the defendant (Uniden) had infringed plaintiff's (EFJ's) computer program, which EFJ had developed for use in a mobile logic trunked radio (LTR) system.[103] Although Uniden had reverse engineered the LTR code in an attempt to create its own version of code compatible with the radios in EFJ's LTR system, the court found compelling evidence of substantial similarity between defendant Uniden's code and EFJ's LTR code.[104] In holding the defendant Uniden liable for infringement, the court nevertheless condoned the practice of decompilation by noting that:

> [t]he mere fact that defendant's engineers dumped, flow charted, and analyzed plaintiff's code does not, in and of itself, establish pirating. As both parties' witnesses admitted, dumping and analyzing code is a standard practice in the industry. Had Uniden contented itself with surveying the general outline of the EJF program, thereafter converting the scheme into detailed code through its own imagination, creativity, and independent thought, a claim of infringement would not have arisen.[105]

By recognizing the importance of reverse engineering, the *E.F. Johnson* decision reflects an important leap towards legitimizing decompilation of copyrighted computer programs. This trend continued in *NEC v. Intel*.[106] In *NEC*, Hiroaki Kaneko, a software engineer at NEC, reverse engineered the microcode used

---

102. 228 U.S.P.Q. (BNA) 891 (D. Minn. 1985).

103. *Id.* at 893.

104. *Id.* at 896. The court cited a number of facts as evidence of substantial similarity: (i) the Uniden code made use of the same sampling rate as used by EFJ, even though the Uniden code was to operate with a newer and faster microprocessor that was capable of using faster sampling rates; (ii) the Uniden code used an error sample technique that was identical to the one used in EFJ code, even though this technique was efficient for the EFJ's code but inefficient when used in the Uniden code; (iii) the Uniden code contained an "H-matrix" for error detection, which was an exact duplicate of the one use by EFJ, even though thirty-one other matrices were available.; (iv) the Uniden code contained an inverse H-matrix identical to the inverse H-matrix found in the EFJ code; (v) both codes contained identical superfluous instructions of duplex operation, even though duplex transmissions were determined to be infeasible after the EFJ code was written, but before the Uniden code was "written;" and (vi) both codes contained an identical error in the codes' "select call prohibit" feature, which was intended to cause a busy signal in certain situations. *Id.* at 896-99.

105. *Id.* at 903 n.17.

106. 10 U.S.P.Q.2d (BNA) 1177 (N.D. Cal. 1989).

in Intel's 8086/88 microprocessors in order to develop a version of the microcode for NEC's V20 and V30 microprocessors, which were comparable to Intel's 8086/88 microprocessors.[107] In so doing, Kaneko decompiled Intel's microcode and studied the decompiled code in hopes of creating a new version of the microcode.[108] Based on the preliminary version of the decompiled code (Rev.O), Kaneko subsequently created another version of the microcode (Rev.2), which was the final version used in NEC's V20 and V30 microprocessors.[109]

Intel claimed that the Rev. 2 microcode infringed its copyright in the 8086/88 microcode based on several similarities between the two programs and other indications of copying.[110] For instance, both programs made use of similar "patches" written to overcome a "bug" in their respective microprocessors[111] as well as a number of other nearly identical microsequences.[112] In addition, Kaneko failed to take advantage of the superior features of the NEC V-series microprocessor hardware and, instead, created a program similar in operation to Intel's microcode.[113] Notwithstanding such similarities, the court determined that the Rev. 2 code did not infringe Intel's copyright in its 8086/88 microcode, concluding that Rev. 2, "when considered as a whole, is not substantially similar to the Intel microcode within the meaning of the copyright law."[114] In reaching this conclusion, the court explained that Kaneko had substantially changed the decompiled code (Rev. O) to create the final version (Rev. 2), and thus, "an ordinary observer, considering Rev. 2 as a whole, would not recognize it as having been taken from the copyrighted source."[115]

---

107. *Id.* at 1188. The reproduction of any *hardware* components of the 8086/88 microprocessors was permitted under license. *Id.* The right to duplicate the microcode, however, was not granted under the license. *Id.*

108. *Id.* at 1185.

109. *Id.*

110. *See NEC*, 10 U.S.P.Q.2d (BNA) at 1184-89 (discussing the arguments put forth by Intel in support of its contention that Kaneko created Rev. 2 by copying substantial portions of the 8086/88 Microcode).

111. *Id.* at 1185. In adapting its microcode for use with the 8088 microprocessor, Intel was required to write a patch to overcome a bug in the 8088's interrupt sequence. *Id.*; *see also supra* note 87 (defining "patch"). Such a patch was not required for the microcode when used with the 8086 microprocessor. *NEC*, 10 U.S.P.Q.2d (BNA) at 1185. Likewise, Kaneko wrote a similar patch in creating the interrupt sequence for the V20 microprocessor, although the patch was not required for the V30 microprocessor. *Id.*

112. *See id.* at 1186-87. Both programs used specific sequences that handled errors and "RESET" functions in the same ways. *Id.*

113. *See id.* at 1187-88. Kaneko made minimal usage of a superior "dual bus" capability used for data transfer and could have made more efficient use of available memory. *NEC*, 10 U.S.P.Q.2d (BNA) at 1187-88.

114. *Id.* at 1183. The court did not rule on the legality of the decompiled code itself. *Id.* at 1186.

115. *NEC*, 10 U.S.P.Q.2d (BNA) at 1184. The court made clear that "a defendant

Any remaining similarities, the court reasoned, were due to hardware constraints, which dictated that certain instruction sequences and program operations be accomplished in identical or substantially similar fashion.[116] In support of its reasoning, the court made repeated references to an independent version of comparable microcode that had been created in a "clean room," which contained many of the same similarities.[117] As a result, the court held that NEC's Rev. 2 did not infringe Intel's 8086/88 microcode.[118]

The *E.F. Johnson* and *NEC* decisions illustrate the courts' trend towards encouraging reverse engineering and decompilation of copyrighted computer programs. In particular, the approaches adopted in these cases indirectly condone decompilation and intermediate copying of computer programs, provided the final product

---

may legitimately avoid infringement by intentionally making sufficient changes in a work which would otherwise be regarded as substantially similar to that of the plaintiff." *Id.* at 1187 (citations omitted). Thus, the court found it of no consequence that Kaneko may have referred to the decomplied code as he developed the final version, stating, for example:

> [l]et us assume . . . that when Mr. Kaneko faced the task of writing [specific] microsequences for Rev. O he sought to recall how Intel had handled this difficult problem, or even referred to this character string to make that determination. Let us assume also . . . that he directly copied what he had learned into Rev. O. If he had stopped there . . . a difficult question would arise as to whether what he had taken from the disassembled 8086/88 constituted the technical "idea" for a solution or the "expression" thereof (citing 17 U.S.C. sec. 102(b); *Whelan*, 797 F.2d at 1234). However, . . . Mr. Kaneko changed the subject microsequences substantially in writing Rev. 2 [and] . . . there remains no basis for a claim of copying or even of substantial similarity.

Id. at 1186-87.

116. *Id.* at 1186-87. Because NEC was permitted to study and duplicate Intel's hardware designs, the court was inclined to permit usage of the accompanying microcode. *Id.*

117. *Id.* at 1188-89. A "clean room" refers to a method of creating software, intended to be comparable to an existing program, where the creation occurs without access to the existing program. *See* Contreras et al., *NEC v. Intel: Breaking New Ground in the Law of Copyright*, 3 HARV. J.L. & TECH. 209, 213 n.23 (1990). The existing program is analyzed to extract its ideas and functions from which specifications may be created. These specifications are then given to programmers in the so-called "clean room," who have not had access to the existing program. The programmers subsequently develop a comparable version of the program based on the specifications. Consequently, similarities between the existing program and the "clean room" program help show that such similarities were dictated by hardware constraints, functional concerns and the like. *See* Miller, *supra* note 10, at 1025.

In *NEC*, once the threat of litigation ensued, an independent software developer was hired by NEC to develop a clean room version of the Intel microcode. *NEC*, 10 U.S.P.Q.2d (BNA) at 1185. This clean room code was substantially similar to certain aspects of both the NEC and Intel microcodes and, as a result, was persuasive evidence of noninfringement. *Id.* at 1184-85.

118. *Id.* at 1190

is not substantially similar to the decompiled code.[119]

B. The Ninth Circuit's Solution: Decompilation as Fair Use

1. The Atari *Case*

In *Atari Games Corp. v. Nintendo of America, Inc.*,[120] the Court of Appeals for the Federal Circuit, applying Ninth Circuit law, sought to justify the practice of decompilation and intermediate copying of computer programs as a fair use. In so doing, the Federal Circuit stated that "reverse engineering object code to discern the unprotectable ideas in a computer program is a fair use," provided the use involves "an authorized copy of [the program]."[121]

In *Atari*, the defendant Atari had created video game cartridges for the Nintendo Entertainment System (NES), which plaintiff Nintendo claimed infringed the copyright in Nintendo's "10NES" program.[122] Atari reverse engineered Nintendo's chips by chemically removing layers from the chips and, through microscopic examination, transcribing the 10NES object code into a handwritten list of ones and zeros.[123] Subsequently, these ones and zeros were fed into a decompiler to produce source code.[124] Atari, however, could not decipher the code, and thus, resorted to other tactics.[125] In particular, Atari's counsel obtained an unauthorized copy of the source code from the U.S. Copyright Office by falsely alleging that the copy was needed for on-going litigation.[126]

Based on these facts, the Federal Circuit analyzed Atari's "reverse engineering" activities under the fair use doctrine. The Federal Circuit's analysis, however, was simplified by the fact

---

119. Both the *E.F. Johnson* and *NEC* cases do not directly provide a justification for the legality of the decompiled code under copyright law, indicating that the courts are willing to "tolerate infringement by the initial decompiled [code]." Contreras, *supra* note 117, at 217-18. One case proposed to justify the practice under § 117(1), where the "copy *was* 'created as an essential step in the utilization' of [the] program." Vault Corp. v. Quaid Software Ltd., 7 U.S.P.Q.2d (BNA) 1281 (5th Cir. 1988)(quoting 17 U.S.C. § 117(1) (1988)). The use of § 117, however, to justify decompilation has repeatedly been criticized. *See, e.g.*, Miller, *supra* note 10, at 1025; *see also* Sega Enter. Ltd. v. Accolade Inc., 24 U.S.P.Q.2d (BNA) 1561, 1568.

120. 24 U.S.P.Q.2d (BNA) 1015 (9th Cir. 1992)

121. *Id.* at 1023-24. The appeal was heard in the Federal Circuit because a claim for patent infringement was raised in the district court proceeding. *Id.* at 1017.

122. *Id.* at 1017-18. The 10NES program is the security system software designed by Nintendo to prevent the NES from accepting unauthorized cartridges. *Id.* at 1017.

123. *Atari*, 24 U.S.P.Q.2d (BNA) at 1017.

124. *Id.*

125. *Id.*

126. *Id.* at 1018

that Atari possessed a purloined copy of Nintendo's code.[127] Indeed, "[b]ecause Atari was not in authorized possession of the Copyright Office copy of 10NES, any copying or derivative copying of 10NES source code from the Copyright Office does not qualify as a fair use."[128] Nevertheless, the Federal Circuit went on to state that, under Ninth Circuit law, "[r]everse engineering, untainted by the purloined copy of the 10NES program and necessary to understand 10NES, is a fair use."[129]

2. *The* Sega *Case*

In *Sega Enterprises, Ltd. v. Accolade, Inc.*,[130] the Ninth Circuit addressed the legality of reverse engineering, and thus, the Federal Circuit's reasoning in *Atari*. The Ninth Circuit recognized the importance of decompilation in the software industry and, like the Federal Circuit in *Atari*, justified the practice as a fair use.[131]

In *Sega*, the plaintiff accused the defendant Accolade of *inter alia* copyright infringement after Accolade had developed video game cartridges compatible with Sega's "Genesis" game console, a device upon which computer video games are played.[132] In order to develop games compatible with the Genesis console, the defendant Accolade performed a two-step process. In the first step of the process, Accolade reverse engineered Sega's video game programs by obtaining commercially available copies of Sega's game cartridges and decompiling the object code stored in the read-only memory (ROM) chips in Sega's games.[133] Accolade engineers studied and analyzed printouts of the decompiled code and loaded the code into a computer for further experimentation.[134] Through this analysis and experimentation, Accolade learned the interface requirements of Sega's code required for compatibility.[135] Based on this knowledge, Accolade created a development manual that

---

127. *Id.*
128. *Atari*, 24 U.S.P.Q.2d (BNA) at 1024.
129. *Id.*
130. 24 U.S.P.Q.2d (BNA) 1561 (9th Cir. 1992).
131. *Id.* at 1562.
132. *Id.* at 1563. Sega develops and markets video entertainment systems, which include the Genesis console and video game cartridges. *Id.* The defendant Accolade is an independent developer, manufacturer and marketer of computer entertainment software, including video games that are compatible with various computer systems such as Sega's Genesis console. *Id.* In addition to the copyright infringement claim, claims were raised for trademark infringement and false designation of origin under sections 32(1) and 43(a) of the Lanham Act, 15 U.S.C. §§ 1114(a)-(1) and 1125(a). *Sega*, 24 U.S.P.Q.2d (BNA) at 1564.
133. *Sega*, 24 U.S.P.Q.2d (BNA) at 1564.
134. *Id.*
135. *Id.*

documented these requirements, although the manual did not include any of Sega's code.[136] In the second step of the process, Accolade developed its own games for the Genesis console, relying on the manual for the interface specifications and without examining other aspects of Sega's code.[137]

The Ninth Circuit held that Accolade's reverse engineering process constituted a fair use.[138] In reaching this decision,[139] the Ninth Circuit's fair use analysis systematically considered the four statutory fair use factors. With respect to the first factor,[140] the Ninth Circuit determined that Accolade's direct purpose for copying Sega's code, and thus, Accolade's direct use of the copyrighted material, was to study the interface requirements of Sega's code for achieving compatibility with the Genesis console.[141] Therefore, even though Accolade's ultimate purpose was the release of Genesis-compatible games for sale, this "commercial aspect of its use can best be described as of minimal significance."[142] Consequently, the Ninth Circuit determined that the first factor weighed in favor of fair use.

The Ninth Circuit determined that the fourth factor,[143] which is closely related to the first factor, also favored Accolade and a fair use determination. In analyzing the likelihood of adverse effects on the market or potential market, the Ninth Circuit determined that many consumers were likely to purchase both games, and that Sega's attempt to monopolize the market was not

---

136. *Id.*

137. *Id.* Subsequently, Accolade learned that Sega was releasing a new version of the Genesis console (Genesis III) that included a trademark security system (TMSS), which required the use of the letters "S-E-G-A" in order to initialize the program. *Id.* at 1564. Accolade reverse engineered a "small segment" of the program in order to learn the initialization code for the program. *Id.* In addition to the reverse engineering issue, this use raised issues of trademark infringement and false designation of origin under sections 32(1) and 43(a) of the Lanham Act, 15 U.S.C. §§ 1114(a)(1) and 1125(a). *Id.*

138. *Sega*, 24 U.S.P.Q.2d (BNA) at 1574.

139. The Ninth Circuit reversed the district court's holding that Accolade's reverse engineering practice was not a fair use. The district court determined that Accolade had performed the reverse engineering steps for the commercial purpose of marketing a competing product, which is presumptively unfair, and that Sega failed to rebut this presumption. *Id.* at 1569. The Ninth Circuit, however, agreed with the district court's other conclusions that such reverse engineering was not permitted under § 102(b) or § 117. *Id.* at 1567-68.

140. The first factor requires inquiry into "the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes." 17 U.S.C. § 107(1) (1990).

141. *Sega*, U.S.P.Q.2d (BNA) at 1570.

142. *Id.*

143. The fourth factor requires consideration of "the effect of the use upon the potential market for or value of the copyrighted work." 17 U.S.C. § 107(4) (1990).

an equitable basis for precluding fair use.[144] Consequently, the Ninth Circuit found that the fourth factor weighed in favor of fair use, even though Sega may suffer a "minor economic loss."[145] As a result, the Ninth Circuit minimized the importance of the commercial aspects and effects of Accolade's process in finding that the first and fourth factors weighed in favor of fair use.

Subsequently, the Ninth Circuit completed its fair use analysis by evaluating the other two factors. With respect to the second factor,[146] the Ninth Circuit found that, because computer programs are disseminated in object code form, the nature of the work favored fair use in order to access the unprotected ideas embodied in the program.[147] Finally, the Ninth Circuit determined that the third factor weighed against fair use since the entire work was copied; however, the Ninth Circuit accorded this factor little weight since the ultimate use was limited.[148] Upon weighing these factors, the Ninth Circuit concluded that "where disassembly [or decompilation] is the only way to gain access to the ideas and functional elements embodied in a copyrighted computer program, and where there is a legitimate reason for seeking such access, disassembly or decompilation is a fair use of the copyrighted work as a matter of law."[149]

C. *The Need for a Better Solution?*

1. *The Problems With Fair Use*

The *Atari* and *Sega* cases indicate that courts are willing to permit decompilation of computer software under limited circumstances. These cases, however, stretch the fair use doctrine to its extreme, and perhaps beyond.[150] In particular, the courts' fair use analysis in these cases is unpersuasive and illustrates the difficulty in permitting decompilation under current copyright law. For example, the *Atari* court's analysis of the fair use factors was simplified by the fact that Atari had illegally obtained a copy of the source code at issue. This fact allowed the Federal Circuit to dispose of Atari's fair use defense with relative ease since Atari was in possession of an unauthorized copy of the source code.[151]

---

144. *Sega*, 24 U.S.P.Q.2d (BNA) at 1571.

145. *Id.*

146. The second factor involves evaluating "the nature of the copyrighted work." 17 U.S.C. § 107(2) (1990).

147. *Sega*, 24 U.S.P.Q.2d (BNA) at 1571. The Ninth Circuit made clear, however, that such access only favors fair use if there are no other means to gain access to the ideas, such as by viewing the screen display. *Id.* at 1572 n.7.

148. *Id.* at 1573.

149. *Id.* at 1574.

150. *See supra* notes 117-46 and accompanying text.

151. *See Atari*, 24 U.S.P.Q.2d (BNA) at 1024.

Thus, the *Atari* case's analysis of fair use and its relation to decompilation can be considered "pure dictum."[152]

The *Sega* court's analysis of the fair use factors is more comprehensive than the analysis in *Atari*, although similarly unpersuasive. Although the *Sega* court systematically considered each of the four statutory factors, the *Sega* court's analysis improperly minimizes the obvious commercial nature and purpose of the copying, and thus stretches the fair use doctrine too far.[153] Such a distortion of the fair use doctrine is not in harmony with the intent of Congress. For example, in enacting legislation to protect semiconductor chip design, Congress determined that the fair use provisions under copyright law were inappropriate to accommodate reverse engineering.[154] Consequently, Congress chose to permit reverse engineering through a "specific provision closely tailored to the needs of the semiconductor industry, rather than through an extension of the fair use doctrine."[155] Congress did this specifically in order to avoid creating a general reverse engi-

---

152. *See* Miller, *supra* note 10, at 1015 n.181. Specifically, Miller suggests that, because *Atari* was in possession of an unauthorized copy of the source code, "much of the opinion's discussion of the fair use doctrine is pure dictum." *Id.*

153. The *Sega* court seemed determined to justify the limited decompilation of copyrighted computer software, using the fair use doctrine as the vehicle to reach such a result. In analyzing the first statutory factor, the *Sega* court recognized that "the fact that copying is for a commercial purpose weighs against a finding of fair use. . . . However, the presumption of unfairness that arises in such cases can be rebutted by the characteristics of a particular commercial use." *Sega*, 24 U.S.P.Q. (BNA) at 1569 (citations omitted). The court went on to nevertheless conclude that "although Accolade's ultimate purpose was the release of Genesis-compatible games for sale, the direct purpose in copying Sega's code, and thus its direct use of the copyrighted material, was simply to study the functional requirements for Genesis compatibility so that it could modify existing games and make them usable with the Genesis console." *Id.* at 1570. Thus, the court held that "any commercial 'exploitation' was indirect or derivative . . . [indicating that the copying] was for a legitimate, essentially non-exploitative purpose, and that the commercial aspect of [Accolade's] use can best be described as of minimal significance." *Id.* This analysis is difficult to accord with the Supreme Court's view in *Sony* that such commercial use creates a strong presumption of unfairness.

Similarly, in analyzing the fourth statutory factor, the *Sega* court concluded that the market impact of the use weighed in favor of a finding of fair use. The court reasoned that any harm to Sega would be limited since consumers are likely to buy multiple game cartridges. *Id.* at 1571. This analysis has been criticized based on an inappropriate definition of the relevant market. *See* Miller, *supra* note 10, at 1020 (criticizing the market defined in terms of Sega's game cartridges and suggesting a different result had the market been defined in terms of home entertainment systems, where Sega and Accolade are fierce competitors).

154. *See* Stern, *Determining Liability for Infringement of Mask Work Rights Under the Semiconductor Cip Protection Act*, 70 MINN. L. REV. 217, 329 (1985); Leo J. Raskind, *Reverse Engineering, Unfair Competition, and Fair Use*, 70 MINN. L. REV. 385, 393 (1985); Miller, *supra* note 10, at 1024; *see also infra* notes 162-78 (discussing the protection of such semiconductor chip designs).

155. *See* Miller, *supra* note 10, at 1024.

neering privilege under the fair use doctrine.[156] Thus, "[g]iven this Congressional intent, the *Sega* court's judicial creation of even a circumscribed reverse engineering privilege under the fair use doctrine seems singularly inappropriate."[157] Therefore, decompilation of copyrighted computer programs should be justified on grounds other than fair use.

2. *The Limitations of § 102(b)*

Section 102(b) of the Copyright Act is relevant to decompilation, but is insufficient to justify decompilation. Section 102(b) codifies the fundamental principle of copyright law that a copyright in a work does not extend protection to the ideas embodied in the work, but instead protects only the expression of such ideas.[158] Therefore, others may copy and use the ideas in a copyrighted work, provided the author's expression is not likewise appropriated. Although this principle provides some justification for access to ideas embodied in a work, it does not justify the copying of *both* protected and unprotected elements of the work.[159] As a result, § 102(b) provides insufficient grounds for justifying the decompilation of computer programs.

3. *The Limitations of § 117*

Like § 102(b), § 117 of the Copyright Act is an insufficient basis for permitting decompilation.[160] Section 117 allows owners of computer programs to use their programs and place copies of their programs into their computer's memory without being subjected to potential liability for copyright infringement.[161] In general, § 117 is a narrow exception for copying a computer program when such copying is essential to the legitimate use of the original program. Due to its limited scope, the weight of authority on the subject indicates that § 117 does not protect someone who decompiles or otherwise makes copies of a program for reverse engineering purposes.[162]

---

156. *Id.*

157. *Id.*

158. *See supra* notes 22-33 and accompanying text (describing the idea/expression dichotomy codified in 17 U.S.C. § 102(b)).

159. *See* Miller, *supra* note 10, at 1016 (citing *Atari* for implicitly rejecting this justification for decompilation).

160. *See supra* note 52 (reciting 17 U.S.C. § 117 (1990)).

161. *Id.*

162. *See supra* note 51 (discussing narrow scope of section 117).

4. Summary: The Inadequacy of the Present Statutory Framework

The present statutory framework of the Copyright Act is inadequate to accommodate the practice of reverse engineering and decompilation of copyrighted computer programs. Although the fair use doctrine has recently been stretched to cover this practice, the doctrine is inappropriate to support such a practice. Likewise, the idea/expression dichotomy in § 102(b) is insufficient to recognize a decompilation privilege. Similarly, § 117's narrow exceptions for archival, adaptions, and the like do not provide adequate grounds for permitting decompilation. Consequently, the present statutory framework, properly construed, does not permit decompilation and, therefore, requires revision to address this situation.[163] As a result, the present statutory framework must be modified in order to properly accommodate reverse engineering and decompilation.[164]

V. A PROPOSED SOLUTION: LEGITIMIZING DECOMPILATION BY STATUTE

A. The Semiconductor Chip Protection Act (SCPA) as a Guide

1. Computer Programs: The Analog of a Semiconductor Chip?

A semiconductor chip is a fingernail-sized wafer of silicon, which may contain more than a million electronic elements.[165]

---

163. Other considerations compel the conclusion that the present statutory framework should be modified to accommodate decompilation, rather than stretching the current provisions to permit the practice. For instance, CONTU specifically considered those circumstances where someone in lawful possession of a program might legally modify or copy that program. *See* Miller, *supra* note 10, at 1023. CONTU subsequently recommended that section 117 be enacted to permit copying the program for archival or, where essential to the program's use, copying or adapting the program. *Id.* This suggests that Congress did not wish to extend the copyright provisions which were current at that time to accommodate the peculiarities of computer programs.

164. The European Community (EC) software directive includes an Article that explicitly permits decompilation to achieve interoperability. *See Amended Proposal for a Council Directive on the Legal Protection of Computer Programs*, 33 J.O. COMM. EUR. (No. C 320) 22, 25 (1990). Under Article 6(1)(c) of the directive, the right of decompilation is limited to instances where the decompilation is "indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs." *Id.* Article 6(2)(c) further states that the decompiled code may not be "used for the development, production or marketing of a computer program substantially similar in its expression, or for any other act which infringes copyright." *Id.*

165. *See* John A. Kidwell, *Software and Semiconductors: Why are We Confused?*, 70 MINN. L. REV. 533, 541 (1985). Technically, these electrical elements are configured in three dimensions. *See* Brooktree Corp. v. Advanced Micro Devices Inc., 24 U.S.P.Q.2d (BNA) 1401, 1404 (Fed. Cir. 1992). The elements are built in layers

The layout of the chip requires careful selection and configuration of these elements and their connections in order to achieve the chip's desired functions.[166] The chip design and layout are, in most cases, driven by the function that the chip, or some portion thereof, is sought to perform.[167] Thus, although the chip is typically thought of as hardware, or something in between hardware and software, the chip has many similarities to a computer program.[168] For example, the chip has an architecture or logic that is analogous to instructions that direct the operation of the elements embodied on the chip.[169] These instructions are, therefore, similar to the lines of code in a computer program, which likewise direct the operation of the computer.

Moreover, both chips and programs are created in similar ways. Chips are usually created by diagramming the general layout of the chip design, testing the design through computer simulation and then reducing the design to circuit form during the chip fabrication process.[170] Similarly, computer programs are created by writing a flowchart that gives an overall impression of the structure of the program, coding the program itself to create the program's source code, compiling the program and running and testing the program on the computer.[171]

Due to these and other similarities, it can be argued that chips and programs should not be treated differently.[172] Both types of technology have utilitarian aspects about them, which makes the application of copyright principles troublesome.[173] Since CONTU has recommended that computer programs be protected by copyright, exceptions to the copyright law are the proper means in which to balance issues raised by the utilitarian aspects of computer programs.[174] Because of the inadequacies of the pre-

---

using a series of "masks" which are produced using photographic depositing and etching techniques. *Id.* In this way, layers of metallic, insulating and semiconductor material are deposited in the desired patter on each wafer of silicon. *Id.*; *see also* 17 U.S.C. § 901(a) (defining "mask work" and "semiconductor chip product").

166. *See* Kidwell, *supra* note 165, at 541.

167. *Id.* at 543.

168. Notwithstanding these similarities, semiconductor chips are protected by their own distinct legislation, which is based, in part, on copyright principles as well as other principles tailored solely to the *uniqueness* of the semiconductor ships. *See* 17 U.S.C. §§ 901-914 (1990); *see also Brooktree*, 24 U.S.P.Q.2d (BNA) at 1404.

169. *See* Kidwell, *supra* note 165, at 541.

170. *Id.* at 543.

171. *Id.*

172. *See, e.g.*, Samuelson, *CONTU Revisited*, *supra* note 11, at 727-49; *see generally* Kidwell, *supra* note 165; Samuelson, *Sui Generis Protection*, *supra* note 15.

173. *See* Samuelson, *Sui Generis Protection*, *supra* note 15, at 473. Utilitarian works are not subject to copyright protection. See 17 U.S.C. § 101 (1990) (defining "useful article").

174. See *supra* notes 13-15 and accompanying text for a discussion of CONTU

sent statutory framework,[175] reverse engineering and decompilation of computer programs should be addressed by such means.

2. *Reverse Engineering Under SCPA*

The inherent similarities between semiconductor chips and computer programs suggest that the two types of technology should be protected, in part, by an analogous statutory approach.[176] Although the protection scheme developed for semiconductor chips (entitled the Semiconductor Chip Protection Act or SCPA) includes a number of similarities to the copyright principles that protect computer programs, this chip protection scheme includes several significant differences from traditional copyright law.[177] Of these differences, the specific statutory exception for reverse engineering is most notable.

Under the SCPA, reverse engineering is a statutory defense that allows a chip producer to engage in reverse engineering and, in the process, be excused from infringement, provided the end product is itself original.[178] In performing such reverse engineering, the producer may disassemble, study and analyze an existing chip in an effort to learn its flow and organization.[179] Subse-

---

recommendations.

175. See *supra* notes 150-62 and accompanying text for a discussion of problems with fair use and the inadequacies of the present statutory framework.

176. *See generally* Samuelson, *Sui Generis Protection*, *supra* note 15.

177. *See* 17 U.S.C. §§ 901-914 (1990); *see also Brooktree*, 24 U.S.P.Q.2d at (BNA) 1404-5. In general, the Semiconductor Chip Protection Act, 17 U.S.C. §§ 901-914 (1990), grants "certain exclusive rights to owners of registered mask works, including the exclusive right 'to reproduce the mask work by optical, electronic, or any other means,' and the exclusive right 'to import or distribute a semiconductor chip product in which the mask work is embodied.'" Brooktree, 24 U.S.P.Q.2d (BNA) at 1405 (quoting 17 U.S.C. § 905). "Mask works that are not 'original,' or that consist of 'designs that are staple, common-place, or familiar in the semiconductor industry, or variation of such designs, combined in a way that, considered as a whole, is not original,' are excluded from protection." *Id.* (quoting 17 U.S.C. § 902(b)). Like copyright law, chip protection also does not extend to "any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is embodied." 17 U.S.C. § 902(c).

178. *See Brooktree*, 24 U.S.P.Q.2d (BNA) at 1407. In particular, the SCPA provides that it is not infringement of a registered mask work for:

> (1) a person to reproduce the mask work solely for the purpose of teaching, analyzing, or evaluating the concepts or techniques embodied in the mask work or the circuitry, logic flow, or organization of the components used in the mask work; or
>
> (2) a person who performs the analysis or evaluation described in paragraph (1) to incorporate the results of such conduct in an original mask work which is made to be distributed.

17 U.S.C. § 906(a).

179. *See Brooktree*, 24 U.S.P.Q.2d (BNA) at 1407.

quently, the developer may use any knowledge gained during this study and analysis to create an *original* chip.[180] In developing the new chip, the developer can create a new design that implements the same or equivalent functions as the existing chip, so long as a different design layout is used to achieve this implementation.[181] Such chip development will almost always result in a "paper trail" of documentation that can be used to distinguish between a substantial copy and the product of legitimate reverse engineering.[182] Because of the similarities between chips and programs, the SCPA's reverse engineering exception can be adapted for use with computer programs.

B. Shaping A Limited Decompilation Exception

1. The New Statutory Framework

This Article proposes a limited decompilation exception that permits the decompilation of copyrighted computer programs by statute. This exception codifies the approach taken in *NEC v. Intel*[183] and permits a developer to decompile a copyrighted computer program for study and analysis in order to gain access to the program's ideas and other unprotected elements. The exception is not available when such ideas and elements can be gleaned from other sources such as program manuals, flow charts, interface specifications or screen displays. The developer may subsequently incorporate the program's ideas, the unprotected elements, as well as elements necessary for compatibility, into a "new" version of the program, provided that the final product is not, when considered as a whole, substantially similar to the decompiled code. By permitting the use of these ideas and ele-

---

180. *Id.*

181. *Id.* The legislative history to the SCPA elaborates on the requirements for a derivative design not to be an infringement:

> The end product of the reverse engineering process is not an infringement, and itself qualifies for protection under the Act, if it is an original mask work as contrasted with a substantial copy. If the resulting semiconductor product is not substantially identical to the original, and its design involved significant toil and investment, so that it is not mere plagiarism, it does not infringe the original chip, even if the layout of the two chips is, in substantial part, similar.

Id. at 1408 (citing *Explanatory Memorandum — Mathias-Leahy Amendment to S. 1201*, 130 CONG. REC. S12, 916 (daily ed. Oct. 3, 1984)).

182. The paper trail may include circuit diagrams, preliminary design layouts, computer simulations and the like. Such evidence is thought to distinguish illegitimate and legitimate behavior: the paper trail is "expeted to document efforts in 'analyzing or evaluating the concepts or techniques embodied in the mask work or the circuitry, logic flow or organizations of components used in the mask work,' as the effort required would be reflected in the documents." *Id.*

183. *See supra* notes 108-18 and accompanying text.

ments, the exception presupposes a commercial motive and purpose. Such a decompilation exception might read:[184]

§117A. LIMITATIONS ON EXCLUSIVE RIGHTS: REVERSE ENGINEERING OF COMPUTER PROGRAMS

Notwithstanding the provisions of § 106, it is not an infringement of the exclusive rights of the owner of a copyrighted computer program for:

(1) A person, in rightful possession of an authorized copy of the program, to decompile, disassemble, or otherwise translate the program from machine-readable form to an assembly or higher-level programming language for the purpose of studying, analyzing, or evaluating the ideas, concepts, techniques or unprotected expression embodied in the program; or

(2) A person, who performs the study, analysis or evaluation described in paragraph (1), to incorporate the results of such reverse engineering in a final product made for distribution, provided that the final product is, when considered as a whole, not substantially similar to the copyrighted computer program and cannot be recognized to have been taken from a copyrighted source.

2. Application of the Limited Decompilation Exception

The limited decompilation exception would, as an affirmative defense, place the burden of proof on the accused infringer. The defendant would, therefore, be required to persuade the fact finder that the new version of the decompiled code fits within the limited decompilation exception in order to rebut the plaintiff's prima facie case of infringement.[185] Any similarities in subroutines or other sequences, for example, could be attributed to specific compatibility and hardware constraints. Clean room evidence would be admissible to corroborate any theory of noninfringement, although such evidence is not required.[186] Likewise, "paper trail" evidence can be used to distinguish a substantial copy from a legitimate reverse engineered product. Expert testimony may also prove useful in drawing such a distinction.

---

184. *Cf. supra* note 178 (reciting the reverse engineering provision under the SCPA).

185. This approach has been suggested under the SCPA as well. *See* Stern, *Determining Liability, supra* note 155, at 338.

186. Certainly, the "new" version of decompiled code would not need to be created in a clean room since such a requirement would retard cumulative innovation. *See supra* notes 64-71 and accompanying text for a discussion of cumulative innovation. *See also supra* note 114 (discussing *NEC* and condoning access to the decompiled code during development of a new version of decompiled code).

3. The Benefits of a Statutory Solution Outweigh the Costs

The new limited decompilation exception is carefully tailored so that it permits decompilation without unduly diminishing the protection of computer programs. The exception permits developers and programmers to access ideas and, at the same time, is protective enough to prevent piracy. The proposed exception, by explicitly accommodating decompilation, allows the software industry and consumers alike to benefit from cumulative innovation, standardization, and the creation of compatible programs.

CONCLUSION

This Article has analyzed reverse engineering and the decompilation of computer programs under current copyright law. The analysis has illustrated the struggle that courts have faced in balancing the need to protect computer programs, on the one hand, against the need to accommodate reverse engineering and decompilation, on the other. The analysis has critiqued the courts' solution to date, which seeks to stretch an inadequate statutory framework too far. Consequently, the analysis calls for the creation of a new statutory section that exempts decompilation from the scope of copyright infringement under limited circumstances. In particular, a new limited decompilation exception has been proposed that legitimizes the practice of decompilation by placing it squarely within the provisions of the copyright law.

The proposed decompilation exception is carefully tailored to permit decompilation without unduly diminishing the protection for computer programs. This exception is, therefore, protective enough to prevent piracy and, at the same time, permits developers and programmers to access a program's ideas and unprotected elements. By explicitly accommodating decompilation, the exception allows the software industry and consumers alike to benefit from cumulative innovation, standardization, and the creation of compatible programs. Courts can apply the exception to the practice of decompilation with relative ease and, more importantly, without the need to force a square peg in a round hole.