COMMENTS

LET THE HACKERS HACK: ALLOWING THE REVERSE ENGINEERING OF COPYRIGHTED COMPUTER PROGRAMS TO ACHIEVE COMPATIBILITY

GARY R. IGNATIN†

As the majority of hobbyists must be aware, most of you steal your software.

-Bill Gates, founder of Microsoft Corporation¹

*Calling all your potential future customers thieves is perhaps 'uncool'
marketing strategy.*

-Homebrew Computer Club Member²

INTRODUCTION

Imagine that you are a computer programmer, and you have come up with a great idea. You would like to develop an ingenious program that will take any English document and translate it into Basque. You realize, however, that virtually everyone in the United States uses a program called WriteIt for their word processing. Unless your translator can accept documents created with WriteIt, it will be virtually unmarketable. Making your program compatible will likely be extremely difficult, and you will need to know a great deal of information about WriteIt in order to succeed. To make matters worse, the makers of WriteIt, fearing even the remotest possibility of competition, have refused to extend any support.

† B.S.E. Computer Science 1990, Princeton University; J.D. Candidate 1993, University of Pennsylvania. I wish to thank the comment editors and associate editors for their considerable assistance and input. All errors are of course my own. Also, much gratitude to Manny Pokotilow for his assistance, and to Ellis Klein, Ricky Ignatin, Dan McMurrer, Mike Miracle, Steve Roberts, Matt Walter, and Mom and Dad for vast quantities of moral support. Finally, I wish to dedicate this Comment to Philip Ignatin, who will be a source of inspiration to me long after the memories of Law Review have faded.

¹ PAUL FREIBERGER & MICHAEL SWAINE, FIRE IN THE VALLEY 153 (1984) (quoting Bill Gates's *Open Letter to Hobbyists*, "which was published, among other places, in the 'Homebrew Computer Club Newsletter'").

² MICHAEL MORITZ, THE LITTLE KINGDOM: THE PRIVATE STORY OF APPLE COMPUTER 233 (1984) (quoting a club member's response to Bill Gates).

Will it be possible to develop your Basque translator? The necessary information cannot be obtained by examining the copies of WriteIt that are sold in stores—they are completely unreadable, consisting entirely of millions of zeros and ones that only the computer can understand.³ It is possible to reverse engineer⁴ the WriteIt program, transforming the ones and zeros to a form that is readable by humans. Your lawyers inform you, however, that this reverse engineering would be a very costly copyright infringement. Your dreams of a Basque translator have been unceremoniously dashed, and the world has lost a potentially great and innovative computer program.

While a Basque translator may not be a “must have” program for most users, many other innovative and useful programs are kept from the market because of their inability to achieve compatibility with existing software. The illegality of reverse engineering under current copyright law is a serious obstacle to developers who legitimately desire to create compatible software. This Comment addresses the problem of reverse engineering under current copyright law, and proposes that a “compatibility exemption” be added, allowing reverse engineering for the development of innovative, compatible programs.

Section I briefly describes the technology of computer software and the copyright laws under which it is protected. It then describes the basic dilemma which this Comment attempts to solve: a computer program cannot legally be analyzed because (1) programs cannot be understood by humans unless they are reverse engineered; and (2) reverse engineering is itself a violation of copyright law. Section II examines the scope of protection available for computer software and compares it to the protection available for other copyrighted works and other technologies. It concludes that the current protection of computer software is too strict, because it inadvertently allows the ideas of a program to be hidden. The subsequent effect is a restriction on development in the software industry, particularly in the development of compatible software. Section III discusses the costs and benefits of compatabili-

³ Although users can read the output of programs, the actual instructions that the computer executes are generally stored on a disk as a long series of zeros and ones. This format is called machine code or object code. See *infra* notes 5-9 and accompanying text.

⁴ Reverse engineering is the process of taking a finished product and working backwards in order to discover how it works. See *infra* notes 47-49 and accompanying text.

ty, and concludes that the development of compatible programs is essential to innovation in the computer industry. Section IV examines the problem of piracy, which is the primary fear of most software developers. It describes how the current ban on reverse engineering has been used to protect against piracy and suggests that a total ban causes more problems than it solves. Instead of banning all reverse engineering efforts, the law should instead focus on separating the legitimate development of software from unwanted piracy. Finally, Section V contains a proposal for a solution to the reverse engineering dilemma—a compatibility exemption, derived from copyright’s fair use doctrine, which distinguishes between piracy and desirable efforts to create compatible software.

I. A DESCRIPTION OF THE PROBLEM: COMPUTER PROGRAMS, COPYRIGHT, AND REVERSE ENGINEERING

A. *The Technology of Computer Programs*

A computer program, as defined by copyright law, is “a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result.”⁵ These programs, also known as software, operate as a link between computers and the people using them, allowing users to perform very complex tasks without concern about circuits, chips, and the other mysteries hidden inside the machine. When stored within the memory of a computer, a program is kept as a long string of “ones” and “zeros,” with each number represented as a high or low voltage. Each group of these binary digits is an instruction that causes the computer to perform a very basic task, such as “add” or “move.” Programs in this format are said to be written in “machine language” or “object code.”⁶ Although it is possible for engineers to write software in machine language, the process is extraordinarily difficult and tedious, and virtually never done.⁷ Instead, programs

⁵ 17 U.S.C. § 101 (1988).

⁶ See RAYMOND T. NIMMER, *THE LAW OF COMPUTER TECHNOLOGY* ¶ 1.03[2] (1985). For the most part, object code is unreadable by humans. Some “hard-core” programmers will insist that machine code can be understood with considerable patience and skill. Nonetheless, this Comment will assume that such an attempt is prohibitively time-consuming and difficult and that object code is generally unreadable.

⁷ See Dennis S. Karjala, *Copyright, Computer Software, and the New Protectionism*, 28 JURIMETRICS J. 33, 37 (1987).

are written in easily understood, higher-level languages,⁸ which are later translated into the object code that will operate the computer.⁹ Software written in this format is known as source code.

Within the computer there are a number of layers of software, all working in unison to produce the result desired by the user.¹⁰ At the top of this figurative hierarchy are the application programs ("applications")—the software that describes the precise task that the user wants to accomplish.¹¹ Beneath the applications is the operating system, which coordinates the running of the applications, handles the complex task of storing and retrieving information, and performs many of the standard functions needed by the application.¹² At the bottom of the pile 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.¹³

⁸ Some languages, such as assembly language, are symbolically very close to the machine language which operates the computer. Others, such as BASIC, Pascal, C and FORTRAN, bear more than a faint resemblance to English. See Christopher M. Mislow, *Computer Microcode: Testing the Limits of Software Copyrightability*, 65 B.U. L. REV. 733, 743-44 (1985).

⁹ This translation is itself performed by programs, known as compilers, which will automatically convert each higher-level instruction into the appropriate series of machine language commands. A single BASIC instruction may require dozens of lines of object code to achieve the same result. See, e.g., NIMMER, *supra* note 6, at ¶ 1.03[2].

¹⁰ See Pamela Samuelson, *CONTU Revisited: The Case Against Copyright Protection for Computer Programs in Machine-Readable Form*, 1984 DUKE L.J. 663, 680.

¹¹ Examples of applications include word-processing programs, spreadsheets, painting programs, databases and checkbook managers. Application programs, unlike the other software in the computer, are generally sold separately from the computer itself. See, e.g., Robert Garretson, *What's Hot*, COMPUTERWORLD, Jan. 13, 1992, at 66 (describing new applications available for the Microsoft Windows operating system).

¹² This collection of programs, often referred to as system software, performs many of the mundane tasks required by applications, allowing programmers to write their applications at a higher level of abstraction. For example, the system software may contain a complex program, called SQRT, which will calculate the square root of a number. Programmers may then simply have their program "call" SQRT whenever a square root is required, instead of writing the code themselves. See TRACY KIDDER, *THE SOUL OF A NEW MACHINE* 42 (1981); Karjala, *supra* note 7, at 37.

¹³ See Jorge Contreras et al., *NEC v. Intel: Breaking New Ground in the Law of Copyright*, 3 HARV. J. L. & TECH. 209, 209 (1990); Karjala, *supra* note 7, at 36. Unlike application programs, microcode generally resides permanently inside the computer. See Contreras et al., *supra*, at 210. It consists of a series of programs, each performing a specific machine language instruction. Each line of a microcode program is generally a string of zeros and ones, which will be sent as high or low voltages to the digital circuits of the computer. See *id.* "A microcode 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 macroinstruction set." *NEC Corp. v. Intel Corp.*, 10 U.S.P.Q.2d (BNA) 1177, 1178 (N.D. Cal. 1989).

For each of these layers to work together, the programs in a computer must be compatible. Generally defined, programs are compatible when they can work together or work in place of each other. There are, however, many different types of compatibility, and the refusal to distinguish between these types has been a major stumbling block in the understanding of compatible software.¹⁴ In order to achieve any form of compatibility, programs generally must share the same "interface"—the order and system by which data is stored, retrieved, and output.¹⁵ Some programs will use another program's interface so that their program may "link up" to the other, usually to facilitate the exchange of information. Other programs will use the interface so that the compatible program and the original program will act identically—sending the two programs identical input will produce identical output. Knowledge of a program's interface is essential to the development of any compatible program, regardless of which form of compatibility is sought.

B. *Application of Copyright Law to Computer Programs*

Despite the obviously technical nature of computer programs, copyright law now protects software as a form of literary work.¹⁶ This decision in large part is due to a detailed study made for Congress by the National Commission on New Technological Uses of Copyrighted Works (CONTU), which proposed copyright as a solution to the many problems of protecting software.¹⁷ Copyright

¹⁴ For example, two programs may be compatible if they perform the exact same tasks in the same way, or if they both can access data that is stored in a particular format. Compatibility might refer to the ability of two programs to send information to each other, or to one program's ability to run another program. See *infra* notes 153-163 and accompanying text, for the classification of these concepts into "vertical" and "horizontal" compatibility.

¹⁵ For example, consider a database which contains names and addresses, and a word processing program which prints letters containing these names and addresses. If the two programs are to work together, both must store and retrieve the information in the same order (for instance, first name, last name, street, city, state, zip code). The term "interface" may also refer to the way in which information is entered by or conveyed to the user—this is generally called a "user interface."

¹⁶ Literary works are statutorily defined as "works, other than audiovisual works, expressed in words, numbers, or other 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." 17 U.S.C. § 101 (1988).

¹⁷ See NATIONAL COMM'N ON NEW TECHNOLOGICAL USES OF COPYRIGHTED WORKS, FINAL REPORT (1978) [hereinafter CONTU REPORT].

protection has been upheld for all different forms of software,¹⁸ regardless of how they are stored¹⁹ or whether they are written in object code or source code.²⁰ Computer programs do bear a number of similarities to more traditional literary works: they are written, they are easily duplicated,²¹ and they are the product of creative, intellectual labor. Copyright protection offers a number of advantages as a protection scheme for software, because it can be

There is still fierce academic debate over the appropriateness of copyright as a protection scheme for computer programs. There is considerable belief that copyright law is simply inappropriate as a protection scheme for software, due to its policy against protecting utilitarian works, *see infra* note 88, and its inexperience with the protection of technology. Indeed, many of the problems in today's computer law are an inherent result of these shortcomings. *See, e.g.,* G. Gervaise Davis, III, *Reaching the Limits of Copyright: Protecting Programming Languages, Macros, Formats and Computer Hardware Under the Copyright Laws*, in 10TH ANNUAL COMPUTER LAW INSTITUTE, at 77, 79 (PLI Patents, Copyrights, Trademarks & Literary Prop. Course Handbook Series No. 259, 1988) [hereinafter PLI HANDBOOK] ("To a large degree [the decision to protect software under copyright] predestined today's legal and public policy dilemmas . . ."); Karen Hooten, *Is Copyright Right?*, COMPUTER LANGUAGE, Mar. 1991, at 99, 102 ("The fundamental fault behind the legal confusion is that copyright law has little experience with technology.").

This Comment focuses upon one flaw in copyright protection for software, namely the illegality of reverse engineering; the larger issue of copyright's appropriateness is beyond the scope of this Comment. For criticism of CONTU's decision to employ copyright protection for software, *see, e.g.,* Ronald Abramson, *Why Lotus-Paperback Uses the Wrong Test and What the New Software Protection Legislation Should Look Like*, COMPUTER L., Aug. 1990, at 6; Samuelson, *supra* note 10, at 703-05; Pamela Samuelson, *Creating a New Kind of Intellectual Property: Applying the Lessons of the Chip Law to Computer Programs*, 70 MINN. L. REV. 471, 475 (1985).

¹⁸ *See, e.g.,* Whelan Assoc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222 (3d Cir. 1986) (finding infringement of an application program which efficiently organized a dental laboratory), *cert. denied*, 479 U.S. 1031 (1987); Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983) (protecting the operating system of the Apple II computer), *cert. dismissed*, 464 U.S. 1033 (1984); NEC Corp. v. Intel Corp., 645 F. Supp. 590 (N.D. Cal. 1986) (finding the microcode for Intel's microprocessor copyrightable).

¹⁹ *See Franklin Computer*, 714 F.2d at 1249.

²⁰ *See id.* at 1249; Williams Electronics, Inc. v. Artic Int'l, Inc., 685 F.2d 870, 876-77 (3d Cir. 1982).

²¹ In fact, the ease with which programs may be copied was the primary motivation for CONTU's decision. *See NIMMER, supra* note 6, ¶ 1.03[1]; Raymond T. Nimmer & Patricia A. Krauthaus, *Copyright and Software Technology Infringement: Defining Third Party Development Rights*, 62 IND. L.J. 13, 21 (1987).

attained with minimal cost,²² has a long duration,²³ and it "creates exclusive legal rights to reproduce copies of a 'work.'"²⁴

1. Idea v. Expression

Copyright protection does not, however, extend to every aspect of a copyrighted work. While copyright protects all of an author's expression of a given idea, it does not protect the idea itself. Section 102(b) of the Copyright Act provides: "[i]n no case does copyright protection for an original work of authorship extend to an idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work."²⁵ Everyone is thus free to take and use the ideas from a copyrighted work with impunity, so long as the author's expression remains uncopied. Furthermore, an author cannot copyright the expression of his work if there is only a limited number of ways in which the work's idea can be expressed.²⁶

Enormous difficulties emerge when one tries to apply this doctrine, sometimes called the idea/expression dichotomy, to computer software. What exactly is the idea of a given program? The definition of an idea might be interpreted very narrowly, so that a program's idea would simply be its purpose—for example, to balance a checkbook or to simulate a typewriter. On the other

²² Unlike patent protection, which requires a costly, time-consuming and lengthy application, copyright protection is automatically given to a work as soon as it is "fixed in any tangible medium of expression . . . either directly or with the aid of a machine or device." 17 U.S.C. § 102(a) (1988). Thus, a program is protected by copyright merely by saving it on a disk.

²³ Copyright protection generally extends fifty years beyond the death of the author. See 17 U.S.C. § 302(a) (1988).

²⁴ Nimmer & Krauthaus, *supra* note 21, at 21; see also 17 U.S.C. § 106(1) (1988); NIMMER, *supra* note 6, ¶ 1.06.

²⁵ 17 U.S.C. § 102(b) (1988).

²⁶ This concept is often called the "merger doctrine," because the idea is said to have merged with its expression. This doctrine exists so that an author cannot gain protection for a process or system simply by copyrighting all of its possible forms of expression.

The description of the art in a book, though entitled to the benefit of copyright, lays no foundation for an exclusive claim to the art itself. The object of the one is explanation; the object of the other is use. The former may be secured by copyright. The latter can only be secured, if it can be secured at all, by letters-patent.

Baker v. Selden, 101 U.S. 99, 105 (1880). For an example of the merger doctrine as applied to software, see *infra* note 40 and accompanying text.

hand, the term might be interpreted very broadly, so that the idea includes the sequence and organization of the program, and the expression is the author's specific description of the process in source or object code.²⁷ This issue was addressed in the landmark case of *Whelan Associates v. Jaslow Dental Laboratory, Inc.*,²⁸ in which the Court of Appeals for the Third Circuit held that:

the line between idea and expression may be drawn with reference to the end sought to be achieved by the work in question. In other words, the purpose or function of a utilitarian work would be the work's idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea.²⁹

These definitions of idea and expression completely changed the scope of software protection, extending protection "beyond the programs' literal code to their structure, sequence, and organization."³⁰ A program may therefore infringe a prior work even though its object and source code are completely different from the prior work.

The subsequent case law, however, has not been uniform in its acceptance of *Whelan*. The Fifth Circuit in *Plains Cotton Coop. Ass'n v. Goodpasture Computer Serv., Inc.*³¹ held that when the sequence

²⁷ One can draw the analogy to a recipe. Consider the following two recipes for cookies:

- a) Mix beaten eggs, flour, milk and sugar. Then bake.
- b) Beat the eggs, and then add the flour, milk and sugar. Mix briskly for 3 minutes. Then bake.

The idea of the first recipe may simply be "the baking of cookies," in which case everything else (including the way the cookies are made) is expression. If this is the case, then the second recipe is an infringement, since its sequence and organization is substantially similar to the first recipe. On the other hand, the idea may be the particular method or process for baking the cookies, in which case the expression is simply the way in which the author describes his or her process in words. Under this definition, the second recipe is not an infringement.

²⁸ 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987).

²⁹ *Id.* at 1236 (emphasis omitted).

³⁰ *Id.* at 1248; *see also* SAS Inst., Inc. v. S & H Computer Sys., Inc., 605 F. Supp. 816 (M.D. Tenn. 1985); Stephen J. Davidson, *Reverse Engineering and the Development of Compatible, and Competitive Products Under United States Law*, in PLI HANDBOOK, *supra* note 17, at 407, 429 (describing SAS Institute as holding that "copying the organizational and structural details of a program constituted infringement where the resulting program was essentially a translation of the original work"). *Whelan* relied upon SAS Institute for the principle that a computer program's copyright could extend beyond its literal elements to its structure and organization. *See Whelan*, 797 F.2d at 1239.

³¹ 807 F.2d 1256 (5th Cir.), *cert. denied*, 484 U.S. 821 (1987).

and organization of a computer program is determined largely by market factors, the organizational similarity between the plaintiff's and defendant's programs will not be sufficient to constitute infringement of the plaintiff's work.³² More recently, in *Computer Assocs. Int'l, Inc. v. Altai, Inc.*,³³ a district court in the Eastern District of New York explicitly rejected the *Whelan* test as "inadequate and inaccurate."³⁴ Although the well-reasoned *Altai* opinion offers hope for a more reasoned approach to the idea-expression problem, the effects of *Whelan* have been widespread, making a determination of infringement far more likely when a program uses non-literal elements of pre-existing software.³⁵

2. The Copyrightability of Interfaces

The idea-expression dichotomy has had a direct impact upon the development of compatible programs. Specifically, courts have had to consider whether an interface is copyrightable; obviously, if an interface is protected, then program developers cannot make their software compatible without infringing the copyright in the original program.³⁶ Although one early case held that an interface was an unprotected idea,³⁷ the majority of courts confronting disputes over the copyrightability of interfaces have reached the opposite conclusion.³⁸ In the most recent case in this area, *Lotus Development Corp. v. Paperback Software International*,³⁹ a federal district

³² See *id.* at 1262.

³³ 20 U.S.P.Q.2d (BNA) 1641 (E.D.N.Y. 1991).

³⁴ *Id.* at 1650.

³⁵ See Karjala, *supra* note 7, at 79-80.

³⁶ See *supra* notes 14-15 and accompanying text.

³⁷ See *Synercom Technology, Inc. v. University Computing Co.*, 462 F. Supp. 1003 (N.D. Tex. 1978), (holding that the input format of a computer program was an unprotected idea). Analogizing the program's input format to the figure-H gear-shift pattern of a car, the *Synercom* court noted that "the pattern . . . may be expressed in several different ways . . . [b]ut the copyright protects copying of the particular expressions . . . and does not prohibit another manufacturer from marketing a car using the same pattern." *Id.* at 1013. This decision also shows an implicit acceptance of compatibility—"[i]n *Synercom*, the particular input format was not the only method available for entering data into a structural analysis program. The defendants elected to seek 'compatibility' with *Synercom* since it was a market leader." Nimmer & Krauthaus, *supra* note 21, at 45.

³⁸ See *Manufacturers Technologies v. CAMS, Inc.* 706 F. Supp. 984 (D. Conn. 1989); *Pearl Sys., Inc. v. Competition Elec.*, 8 U.S.P.Q.2d (BNA) 1520 (S.D. Fla. 1988); *Digital Communications Assocs. v. Softklone Distributing Corp.*, 659 F. Supp. 449 (N.D. Ga. 1987); *Broderbund Software, Inc. v. Unison World, Inc.*, 648 F. Supp. 1127 (N.D. Cal. 1986).

³⁹ 740 F. Supp. 37 (D. Mass. 1990).

court in Massachusetts held that, absent merger between idea and expression, all elements of a user interface may be copyrighted.⁴⁰ In *Lotus*, Paperback's "VP-Planner" program had taken the user interface⁴¹ from Lotus' "1-2-3" program, because Lotus 1-2-3 had become the de facto standard of the spreadsheet industry. Although Lotus' actual source or object code was not taken, VP-Planner was independently developed to look and act like the Lotus program. The *Lotus* court decided to extend copyright protection to Lotus 1-2-3's user interface, and subsequently found that the Paperback program infringed that copyright.⁴² This decision greatly increased the scope of protection for computer software and, if it is followed by other courts, may have made the development of compatible software without infringement practically impossible.⁴³

C. *The Problem of Reverse Engineering: Analyzing the Ideas of Computer Programs*

Copyright law, while protecting the rights of authors, allows others freely to study and analyze their works, so that further advances can be made in their field. For example, a director of suspense films can study the works of past masters in order to perfect her ability. She may replay Alfred Hitchcock's films frame-by-frame, analyzing his ideas and his expression of those ideas.

⁴⁰ Examples of this merger include the rotated "L" screen display which is used in virtually every spreadsheet program on the market, and the use of the "+", "-", "*", and "/" symbols to represent addition, subtraction, multiplication and division. See *id.* at 66.

⁴¹ A user interface comprises the way in which a user controls and interacts with the program. It includes, among other things, the organization of the screen, the method used to give instructions, and the way choices are presented to the user. The user interface is often called the "look and feel" of a program. It is undisputed that much thought and creativity goes into the development of a good user interface. See generally Bill Curtis, *Engineering Computer "Look and Feel": User Interface Technology and Human Factors Engineering*, 30 JURIMETRICS J. 51 (1989) (describing how user interfaces are developed).

⁴² The *Lotus* court, however, did not propose a solution to the idea-expression problem of computer software. Instead, Judge Keeton simply stated:

[I]n making the determination of "copyrightability," the decisionmaker must focus upon alternatives that counsel may suggest, or the court may conceive, along the scale from the most generalized conception to the most particularized, and choose some formulation—some conception or definition of the idea—for the purpose of distinguishing between the "idea" and its expression.

Lotus, 740 F. Supp. at 60.

⁴³ For criticism of this decision, see Abramson, *supra* note 17, at 6-8.

Although she cannot duplicate any of Hitchcock's works, she may use his ideas without infringing his copyrights. Similarly, a science fiction writer could peruse the copyrighted works of Isaac Asimov and Arthur C. Clarke, observing their writing styles and using their ideas to create new stories. The same right to analyze and extract ideas is currently extended to all copyrighted works, with the exception of computer software.

Computer software is generally sold in its object code format, which is generally unreadable by humans.⁴⁴ As a result, virtually all of a program's operation is hidden from the user, preventing any efficient and useful analysis of the program. The uncertain definitions of idea and expression are crucial in delineating the magnitude of this problem. If the idea of a program is solely its purpose, as suggested by *Whelan*,⁴⁵ then all of the program's structure, organization, and actual code represents expression which is protected by copyright. Ideas are not being locked up because a program's purpose can be determined without analysis of its code at all. But if the definition of a program's idea encompasses more than its purpose, then the unreadable nature of object code will hide the program's ideas from discovery. This Comment assumes, without creating explicit definitions for idea and expression, that a program's idea extends beyond its mere purpose; thus, at least some of the ideas of a computer program are hidden from view by the unreadability of object code.⁴⁶

⁴⁴ See *supra* notes 6-7 and accompanying text; see also STEVEN LEVY, HACKERS: HEROES OF THE COMPUTER REVOLUTION 19 (1984) ("Staring at a long list of computer commands written as binary numbers—for example, 10011001100001—could make you into a babbling mental case in a matter of minutes.").

⁴⁵ See *supra* notes 28-30 and accompanying text.

⁴⁶ See *Computer Assocs. Int'l v. Altai*, 20 U.S.P.Q.2d (BNA) 1641 (E.D.N.Y. 1991). The *Altai* recognized the errors of the *Whelan* court, referring to its holding as "simplistic," *id.* at 1649, "inadequate and inaccurate," *id.* at 1650, and "fundamentally flawed," *id.* The court correctly recognized that between identifiable ideas (such as the purpose of a program) and identifiable expression (such as the literal object code) there are a great many "levels of abstraction." Under this "abstractions test," one starts with the program's most identifiable expression (the actual text) and gradually moves towards its more general aspects. At some point in this series of abstractions, a level is reached where expression has been replaced by ideas. "This abstractions test would progress in order of 'increasing generality' from object code, to source code, to parameter lists, to services required, to general outline." *Id.* at 1651.

1. Reverse Engineering: The Key to Unlocking the Ideas

Although the ideas of a program cannot be extracted from its object code, all hope is not lost for those who wish to analyze a piece of software. Engineers generally can unlock the secrets of a technology by a process called "reverse engineering," whereby one inspects or takes apart a finished product to discover how it works, often to create a new product using the discovered knowledge. In the field of software, reverse engineering is performed by translating the unreadable object code of a program into source code that may be studied. This process, which can often be completed automatically by special translation programs, is known as "decompilation" or "disassembly."⁴⁷ Although a program may be decompiled fairly quickly, the extraction of useful information from the newly obtained source code can be a very long and arduous task.⁴⁸ Reverse engineering has been used in most fields of high-technology, including software, to analyze the advances made by new products, and is generally accepted as a standard industry practice.⁴⁹

2. The Problem: Reverse Engineering a Computer Program Is an Infringement of Copyright Law

Despite the necessity of reverse engineering in discovering a program's ideas, current copyright law treats reverse engineering as an infringement. This dilemma arises from the particular process of decompiling software. In order to reverse engineer most technologies, there is no need to actually copy the product; one only needs to purchase a number of examples of the product and then

⁴⁷ Not surprisingly, the programs to accomplish this are known as decompilers or disassemblers. The process may often be done in several steps, changing the machine code first to assembly language, and then to a higher-level, more easily understood language. See Mislow, *supra* note 8, at 743 n.42.

⁴⁸ As an analogy consider a handbook, written in Japanese, which describes the intricacies of the Japanese phone system. Even after one translated the work to English, it would still be painstakingly difficult to fully understand the logic and design of the system.

⁴⁹ See *LaST Frontier Conference Report on Copyright Protection of Computer Software*, 30 JURIMETRICS J. 15, 24 (1989) [hereinafter *LaST Frontier Conference Report*] (noting that some courts have begun to take into account the common software industry practice of reverse engineering); Richard H. Stern, *Determining Liability for Infringement of Mask Work Rights Under the Semiconductor Chip Protection Act*, 70 MINN. L. REV. 271, 327 (1985) ("Semiconductor industry representatives testified that it is an established industry practice to make photographs of one another's chips in order to analyze them and design similar chips.").

take them apart.⁵⁰ When a computer program is reverse engineered, however, a number of copyright violations occur in the process. Generally, a copy of the program must first be made in the computer's memory.⁵¹ The copy must then be translated into a source code, a process which may involve the creation of several additional copies of the program.⁵² The final source code version will itself constitute an unauthorized derivative work of the original

⁵⁰ See Clifford G. Miller, *The Proposal for an EC Council Directive on the Legal Protection of Computer Programs*, 12 EUR. INTELL. PROP. REV. 347, 349 (1990) ("For a three-dimensional product, to reverse engineer it, it is a simple matter to buy as many examples as necessary to take apart, inspect, and test without copying anything.").

⁵¹ In order for a computer to execute or act upon a program, it must first copy that program into its memory. Therefore, every time a program is loaded and run, an unauthorized copy is made. Normally, this type of "copying" is excused under § 117(1) of the Copyright Act. Section 117 was added to the copyright laws to ensure that owners would be able to use their programs without infringement, and to ensure that users could make copies for archival purposes. It states:

Notwithstanding the provisions of section 106, 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 (1988) (emphasis added). See generally Robin Michael, Note, *17 U.S.C. § 117: Is the Amendment to the Copyright Act Adequate to Regulate the Computer Software Market?*, 7 COMPUTER/L. J. 227, 239-43 (1986) (arguing that, as applied by courts, § 117 provides inadequate protection of software owners' rights). It is unclear whether loading a program into memory in order to decompile it is proper "utilization of the computer program" as envisioned by § 117. See *infra* note 57 (discussing *Vault Corp. v. Quaid Software Ltd.*, 847 F.2d 255 (5th Cir. 1988)).

Very often, the software manufacturer will include a "shrink-wrap license" with the software, forbidding all but normal use of the program and banning decompilation in particular. See *infra* note 61 and accompanying text. Thus, even if a particularly determined and skilled programmer attempted to read the object code of the program (rather than decompiling), the mere loading of the program into memory would not be exempted by § 117 and would therefore constitute an infringement.

⁵² These copies will exist only inside the memory of the computer and generally will never be seen by humans. Nonetheless, the CONTU report clearly states that unauthorized copies existing in a computer's memory will constitute infringement. See CONTU REPORT, *supra* note 17, at 13.

object code,⁵³ and merely printing the results on paper will result in the production of further unauthorized copies.

The official United States position,⁵⁴ and the conclusion of most scholars,⁵⁵ is that decompilation is an infringement under current copyright law. The case law in this area has done little to alter this conclusion. The Supreme Court has shown a remarkable reluctance to address this problem, as well as many of the other problems of protecting software under copyright, and indeed has never granted certiorari to such a case.⁵⁶ Moreover, no case has ever directly decided the legality of the decompilation process; instead, the courts have generally focused their inquiries upon whether the product created from the discovered information was

⁵³ Copyright law gives authors an exclusive right to make any works based on their original copyrighted work. These are referred to as derivative works. Regardless of whether the source code is treated as an actual copy or as a derivative work, the decompiled software will constitute a copyright violation. See 17 U.S.C. § 106(1)-(2) (1988).

⁵⁴ When the European Community was considering the question of reverse engineering, the United States was consulted to determine its position on unauthorized decompilation:

The US Government's response indicated that the US Copyright Act does not contain provisions explicitly dealing with reverse engineering or decompilation of computer programs. Under US law, copying or reproducing a computer program without permission of the program owner, the response concluded, is not permitted unless excused by the provisions of the fair use doctrine or as a back-up copy under section 117. The response further identified that there were no cases in which fair use had been applied to decompilation.

Robert J. Hart, *Interfaces, Interoperability and Maintenance*, 13 EUR. INTELL. PROP. REV. 111, 113 (1991).

⁵⁵ See, e.g., Hart, *supra* note 54, at 114 (noting that technical infringement will occur whenever obtaining information from the program code); *LaST Frontier Conference Report*, *supra* note 49, at 24 (noting that "computer programs cannot be 'read' [sic] without disassembling or decompilation, a process that may also involve making a copy of some or all of the code"); Chris Reed, *Reverse Engineering Computer Programs without Infringing Copyright*, 13 EUR. INTELL. PROP. REV. 47, 53 (1991) (concluding that the title of the author's own article is a contradiction in terms, both theoretically and practically).

⁵⁶ See, e.g., *Whelan Assocs. v. Jaslow Dental Lab., Inc.*, 797 F.2d 1222, 1238 (3d Cir. 1986) (holding that copyright principles derived from other areas are applicable to computer programs and that the unauthorized copying of such programs is infringement), *cert. denied*, 479 U.S. 1031 (1987). Moreover, it seems increasingly unlikely that such a case will ever reach the Supreme Court, due to the incredible amount of money at stake in such cases. Because a complete loss can be so costly (often the future of an entire business is at stake), more and more important cases are settling. See, e.g., *Dataline*, DATA BASED ADVISOR, May 1991, available in LEXIS, Nexis Library, Currnt File (describing the eventual settlement between Lotus Development Corp. and Paperback Software International).

an infringing work. To the extent that the issue has been reached in dicta, the courts' opinions have largely been hostile to reverse engineering.⁵⁷

⁵⁷ See, e.g., *Whelan*, 797 F.2d at 1248 (finding infringement where program was reverse engineered to write a similar program for a different computer); *SAS Inst., Inc. v. S & H Computer Sys., Inc.*, 605 F. Supp. 816, 833 (M.D. Tenn. 1985) (condemning reverse engineering of a statistical analysis program); *Hubco Data Prods. Corp. v. Management Assistance Inc.*, 219 U.S.P.Q. (BNA) 450, 455-56 (D. Idaho 1983) (holding that reverse engineering to increase the power of plaintiff's program was infringement).

There are, however, three cases which displayed a more sympathetic attitude towards the decompilation process, although they too did not rule on the legality of reverse engineering. In *NEC Corp. v. Intel Corp.*, 10 U.S.P.Q.2d (BNA) 1177 (N.D. Cal. 1989), a software engineer at NEC reverse engineered Intel's microcode, using the decompiled code as a basis for the development of a compatible set of microcode. The court held that the final derivative work was not an infringement, although it never ruled on the legality of the decompilation itself. See *id.* at 1189. One commentator noted:

[A] genuine issue exists as to whether NEC's initial reverse engineering product, Rev. 0, is a prohibited copy. Yet Judge Gray did not address the issue of whether NEC's Rev. 0 infringed the Intel microcode. Instead, he focused solely on NEC's final product, Rev. 2. As a result, Judge Gray's explicit approval of reverse engineering techniques may itself tolerate infringement by the initial decompiled program (in this case, Rev. 0).

Contreras, *supra* note 13, at 217-18 (footnotes omitted).

The holding of *Intel* seems to recognize reverse engineering as a commonplace and valuable activity. Furthermore, it draws a useful distinction between the act of reverse engineering and use of the discovered knowledge in a subsequent work.

Similar reasoning was used in *E.F. Johnson Co. v. Uniden Corp.*, 623 F. Supp. 1485 (D. Minn. 1985), in which the defendant had reverse engineered code necessary for the development of compatible radios. See *id.* at 1490. Although the defendant was found guilty of infringement the court noted that:

The 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 competitors' codes is a standard practice in the industry. Had Uniden contented itself with surveying the general outline of the EFJ program [rather than copying parts of the code itself] . . . a claim of infringement would not have arisen.

Id. at 1501 n.17.

Finally, in *Vault Corp. v. Quaid Software Ltd.*, 847 F.2d 255 (5th Cir. 1988), Vault made a copy-protection program (PROLOK) which prevented the duplication of any program placed on a PROLOK diskette. See *id.* at 256-57. Quaid reverse engineered PROLOK in order to create a new program (RAMKEY) that would undo the effects of the copy protection, leaving the disks unprotected. The Fifth Circuit, in a decision remarkably favorable towards reverse engineering, held that the copying that occurred during the decompilation of PROLOK was exempted by § 117(1), see *supra* note 51, because the "the copy . . . was 'created as an essential step in the utilization' of Vault's program." *Vault*, 847 F.2d at 261 (quoting 17 U.S.C. § 117(1) (1988)). The court noted that "[s]ection 117(1) contains no language to suggest that the copy it permits must be employed for a use intended by the copyright owner." *Id.* Furthermore, the court struck down a Louisiana law permitting manufacturers, through "shrink-wrap licenses," see *infra* note 61 and accompanying text, to forbid any

The process of decompiling software, therefore, is likely an infringement of copyright. Because the discovery of a program's ideas can only be accomplished through such decompilation, the end result is that its ideas are inadvertently protected under copyright. There is, however, no public policy for allowing the ideas of software, and no other copyrighted works, to remain hidden. Rather, this protection exists solely because: (1) computer programs, unlike all other copyrighted works, can be sold in object code form that will fulfill their designed purpose while remaining indecipherable to humans; and (2) decompilation, the only way to understand such object code, constitutes a copyright violation.

II. THE NEED FOR REVERSE ENGINEERING—THE OVERPROTECTION OF COMPUTER SOFTWARE

As noted above, current copyright law allows developers of software to keep the ideas of their programs hidden. Is this too much protection for computer software? To make this determination, one must look to the goals of copyright law first enumerated in the Constitution.⁵⁸ "[T]he ultimate purpose of copyright legislation is to foster the growth of learning and culture for the public welfare, and the grant of exclusive rights to authors for a limited time is a means to that end."⁵⁹ Because the goal of copyright law is the continued advancement of knowledge, the

copying or modifying of their programs. *See id.* at 270. The holding on this issue was significant because shrink wrap licenses have long been a key weapon in manufacturers' efforts to prevent decompilation.

While the *NEC*, *Uniden*, and *Vault* decisions are encouraging, they unfortunately represent the minority position on the issue of reverse engineering. Unless the Supreme Court explicitly rules that reverse engineering is not an infringement, congressional action will be necessary for decompilation to become legal.

⁵⁸ *See* U.S. CONST. art. I, § 8, cl. 8. ("The Congress shall have Power . . . To 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 . . .").

⁵⁹ REPORT OF THE REGISTER OF COPYRIGHTS ON THE GENERAL REVISION OF THE U.S. COPYRIGHT LAW, 87th Cong., 1st Sess. 5 (Comm. Print 1961). It is settled beyond question that the primary goal of copyright is not to reward authors, but is to promote the advancement of science and the arts. This principle was recently stated in *Sony Corp. v. Universal City Studios, Inc.*, 464 U.S. 417 (1984), where the Supreme Court noted that "[t]he monopoly privileges that Congress may authorize are neither unlimited nor primarily designed to provide a special private benefit. Rather, the limited grant is a means by which an important public purpose may be achieved. It is intended to motivate the creative activity of authors and inventors" *Id.* at 429.

Copyright Act must be tailored in the way that will best promote innovation in the fields it regulates. The issue is therefore whether the inadvertent protection of a program's ideas will unduly burden the advancement of innovation in the software industry. The answer is quite clearly yes.

A. *Copyright Protection of Software as Compared to Other Copyrighted Works*

Copyright protection extends only to a work's expression, leaving the ideas of the work in the public domain.⁶⁰ Because software is distributed in unreadable form, however, the ideas behind a program are impossible to reach. The scope of protection for software is therefore much broader than that of other copyrighted works. Software developers also have the advantage of supplemental contractual protection for their works. It would be ridiculous for an author to stipulate that his book could not be studied, or for a painter to stipulate that his painting could not be analyzed. Software manufacturers, however, routinely use "shrink-wrap licenses" to contractually prevent any unauthorized study or decompilation of their programs.⁶¹

⁶⁰ See 17 U.S.C. § 102(b) (1988).

⁶¹ A manufacturer is limited in the restrictions he may put upon the use of his work after it is sold. See 17 U.S.C. § 109(a), (c) (1988). Therefore, most software "sales" are instead constructed as licenses.

Shrink-wrap licenses are licenses attached to the software package, typically visible through the clear plastic wrapping. A buyer generally "agrees" to adhere to the terms of the license upon the opening of the package. The terms of the license invariably contain a prohibition on copying, modifying, and decompiling the program. See *supra* note 57 (discussing *Vault Corp. v. Quaid Software Ltd.*, 847 F.2d 255 (5th Cir. 1988)), in which the court struck down the use of shrink-wrap licenses to forbid copying or modifying programs). For in-depth discussions of shrink-wrap licenses, and the ramifications of *Vault*, see Deborah Kemp, *Limitations Upon the Software Producer's Rights: Vault Corp. v. Quaid Software Ltd.*, 16 RUTGERS COMPUTER & TECH. L.J. 85 (1990); Page M. Kaufman, Note, *The Enforceability of State "Shrink-Wrap" License Statutes in Light of Vault Corp. v. Quaid Software, Ltd.*, 74 CORNELL L. REV. 222 (1988); D.J. Hale, Recent Development, *Vault Corp. v. Quaid Software, Ltd.: Limits to Copyright Protection for Computer Programs*, 64 TUL. L. REV. 270 (1989).

There is some question as to whether such contractual terms are a form of "misuse." "The patent misuse doctrine was created to deny relief against patent infringement to a patentee 'if he has attempted illegally to extend the scope of his patent monopoly.'" Gerald Sobel, *Recent Developments in Patent Law*, in TECHNOLOGY LICENSING AND LITIGATION 1991, at 7, 89 (PLI Patents, Copyrights, Trademarks, and Literary Property Course Handbook Series No. 308, 1991) (quoting *Dawson Chem. Co. v. Rohm & Haas Co.*, 448 U.S. 176, 180 (1980)). This doctrine has recently been applied to copyright law. See *Lasercomb Am. Inc. v. Reynolds*, 911 F.2d 970, 976 (4th Cir. 1990) (holding that the misuse doctrine applies to copyright as well as patent

Furthermore, unlike all other copyrighted works, computer software can be concurrently protected by trade secret law⁶² because the inner-working of a computer program remains hidden even after its distribution.⁶³ This form of double protection is facilitated by lenient copyright deposit procedures, which allow authors to comply with copyright regulations while keeping the contents of their programs secret; although authors are generally required to deposit a copy of their entire work with the Copyright Office,⁶⁴ software manufacturers need only deposit the first and last twenty-five pages of their program—in unreadable object code.⁶⁵ Trade secret protection also allows manufacturers to license their software longer after copyright protection ends, because the trade secret protection will last until the secret becomes

law); see also Note, *Clarifying the Copyright Misuse Defense: The Role of Antitrust Standards and First Amendment Values*, 104 HARV. L. REV. 1289, 1308 (1991) (advocating a two-step inquiry into First Amendment policy considerations before potential anticompetitive effects under the misuse doctrine); Jere M. Webb & Lawrence A. Locke, Recent Developments, *Intellectual Property Misuses: Developments in the Misuse Doctrine*, 4 HARV. J.L. & TECH. 257, 267 (1991) (advising caution to copyright owners, given the expansion of the misuse doctrine).

⁶² See *infra* notes 75-76 and accompanying text.

⁶³ The Supreme Court has held that trade secret laws and federal intellectual property laws may coexist without preemption. See *Kewanee Oil Co. v. Bicron Corp.*, 416 U.S. 470, 492-93 (1974) (holding that patent law and trade secret law enjoy a peaceful coexistence).

⁶⁴ See 17 U.S.C. § 407(a) (1988).

⁶⁵ The Regulations of the U.S. Copyright Office state:

In cases where a computer program, database, compilation, statistical compendium or the like, if unpublished is fixed, or if published is published only in the form of machine-readable copies (such as magnetic tape or disks, punched cards, semiconductor chip products, or the like) from which the work cannot ordinarily be perceived except with the aid of a machine or device, the deposit shall consist of:

(A) . . . (1) The first and last 25 pages or equivalent units of the source code

37 C.F.R. § 202.20(c)(2)(vii) (1991).

This deposit is used mainly for identification purposes, as opposed to patent's purpose of the public dissemination of information. See Daniel T. Brooks, *Reverse-Engineering Computer Software: Is It Fair Use or Plagiarism?*, in 12TH ANNUAL COMPUTER LAW INSTITUTE 741, 798-800 (PLI Patents, Copyrights, Trademarks & Literary Prop. Course Handbook Series No. 301, 1990). Deposit copies are difficult to find and generally may not be copied. See *id.* at 803. Furthermore, "the head of the reference facilities reportedly has adopted the practice of confiscating notes taken by users of the reference facility [if] they become 'too extensive' in his judgment. In the case of object code, which he can't read, he reportedly confiscates any notes." *Id.*

public knowledge.⁶⁶ Software thus enjoys protection significantly greater than all other copyrighted works.

B. *Protection of Software as Compared to Other Technologies*

1. Comparison With Patented Works

Although computer software is protected by copyright, most technology is protected under patent law. In order to obtain a patent, the inventor must make a complete public disclosure of her work,⁶⁷ describing and illustrating her invention in great detail.⁶⁸ The public has free access to this information, and may use any of the ideas underlying the invention, as long as the invention itself is not appropriated in any way.⁶⁹ This system is in clear contrast to the protection currently provided software, under which the ideas cannot be examined, and thus cannot be taken.

Furthermore, software enjoys all of the benefits that copyrighted works generally enjoy over patented works: longer duration,⁷⁰ low cost of obtaining protection,⁷¹ and lower standards of originality.⁷² Patent protection, unlike copyright protection, also does not

⁶⁶ Normally, the duration of a copyright licensing agreement can be no longer than the duration of the copyright. With trade secret protection, however, the license can be for an unlimited duration. Such licenses, however, may be held to be misuses if they do not distinguish between royalties for the trade secret and royalties for the copyright. See *Boggild v. Kenner Prods.*, 776 F.2d 1315, 1321 (6th Cir. 1985); *Pitney Bowes, Inc. v. Mestre*, 701 F.2d 1365, 1373 (11th Cir.), *cert. denied*, 464 U.S. 893 (1983).

⁶⁷ This requirement is especially significant in the case of an inventor who is not granted the patent. Because her disclosure to the Patent Office is public, she cannot subsequently obtain trade secret protection for her invention. See *Kewanee Oil Co. v. Bicron Corp.*, 416 U.S. 470, 484 (1974); see also *infra* notes 75-76 and accompanying text.

⁶⁸ There are actually two requirements in this disclosure. First, there is an "enabling" requirement, under which the invention must be described in great enough detail so that the average person skilled in the art may recreate the invention. Second, there is a "best mode" requirement, whereby the inventor is required to disclose the invention in the best form currently known. See 35 U.S.C. § 112 (1988). See generally *Sobel*, *supra* note 61, at 13-25 (discussing both disclosure requirements).

⁶⁹ See *Nimmer & Krauthaus*, *supra* note 21, at 18.

⁷⁰ Compare 17 U.S.C. § 302(a) (1988) (stating that copyright protection extends fifty years beyond the life of the author) with 35 U.S.C. § 154 (stating that patent protection lasts seventeen years).

⁷¹ See 17 U.S.C. § 102(a) (1988) (extending copyright protection to original works of authorship as soon as they are "fixed in any tangible medium of expression").

⁷² Compare *Alfred Bell & Co. v. Catalda Fine Arts, Inc.*, 191 F.2d 99, 102 (2d Cir. 1951) (holding that copyright protection necessitates "[n]o large measure of novelty") with 35 U.S.C. §§ 102, 103 (1988) (prescribing that the subject of the patent must be

offer the inventor an exclusive right to prepare derivative works—she receives protection for her invention alone.⁷³ Thus, while the inventor retains a limited monopoly over the use of her invention,⁷⁴ the technological advances which she made are distributed freely to the public.

2. Comparison with Trade Secrets

Computer programs also enjoy more protection than technology protected solely by trade secret law. Trade secrecy is a common-law doctrine which protects against disclosure of certain kinds of information where the value of the piece of information stems directly from its secrecy.⁷⁵ Although the ideas behind a trade secret are not disseminated, reverse engineering is explicitly permitted to allow the discovery of any trade secret.⁷⁶ While

novel and nonobvious).

⁷³ Compare 35 U.S.C. § 154 (1988) (prescribing contents and terms of patent) with 17 U.S.C. § 106(2) (1988) (granting copyright owner exclusive right to prepare or authorize the preparation of derivative works). See also *O'Reilly v. Morse*, 56 U.S. (15 How.) 61, 119 (1854) (holding that the inventor of the telegraph could not claim patent for all devices that used electric current to communicate over long distances; he could only patent the devices he had already invented).

⁷⁴ In the United States, patent protection lasts for seventeen years from the date when the patent issues and extends to the patentee's heirs and assigns. See 35 U.S.C. § 154 (1988).

⁷⁵ The Restatement of Torts states that "[a] trade secret may consist of any formula, pattern, device or compilation of information which is used in one's business, and which gives him an opportunity to obtain an advantage over competitors who do not know or use it." RESTATEMENT OF TORTS § 757 cmt. b (1939).

The Uniform Trade Secrets Act, adopted by 24 states, see *Secure Servs. Tech., Inc. v. Time and Space Processing, Inc.*, 722 F. Supp. 1354, 1359 n.12 (E.D. Va. 1989), defines a trade secret as:

[I]nformation, including a formula, pattern, compilation, program, device, method, technique, or process, that:

(i) derives independent economic value, actual or potential, from not being generally known to, and not being readily ascertainable by proper means by, other persons who can obtain economic value from its disclosure or use, and

(ii) is the subject of efforts that are reasonable under the circumstances to maintain its secrecy.

UNIF. TRADE SECRETS ACT § 1(4), 14 U.L.A. 438 (1990) [hereinafter UTSA].

Because a trade secret may be protected only as long as it is not disclosed, an invention can obviously not be protected by both patent and trade secret law. The disclosure to the patent office will be sufficient to eliminate trade secret protection.

⁷⁶ Trade secrets may be discovered by any proper means:

Proper means include:

....

computer software is afforded all of the benefits of trade secret protection, it does not suffer the main liability of trade secret law—software cannot be reverse engineered, since doing so would violate the program's copyright.

3. Comparison with Semiconductor Chips

Computer software is also protected more strictly than semiconductor chips. The Semiconductor Chip Protection Act of 1984 (SCPA),⁷⁷ which created a separate form of intellectual property⁷⁸ for semiconductor chips,⁷⁹ explicitly included a provision allowing reverse engineering.⁸⁰ Under the SCPA, third parties

2. Discovery by "reverse engineering", that is, by starting with the known product and working backward to find the method by which it was developed. The acquisition of the known product must, of course, also be by a fair and honest means, such as purchase of the item on the open market for reverse engineering to be lawful

UNIF. TRADE SECRETS ACT § 1 cmt., 14 U.L.A. 438 (1990); *see also* SI Handling Sys., Inc. v. Heisley, 753 F.2d 1244, 1262 (3d Cir. 1985) ("It is clear that under Pennsylvania law [following the Restatement definition, plaintiff's technology] is not entitled to trade secret protection if it is susceptible to reverse engineering"); *Secure Serus. Tech., Inc.*, 722 F. Supp. at 1361 (holding that development of a compatible facsimile machine through reverse engineering was legal where the original machine was protected by trade secret); *Acuson Corp. v. Aloka Co.*, 257 Cal. Rptr. 368, 379-80 (Cal. Ct. App. 1989) (noting that reverse engineering is permitted and even encouraged under trade secret law).

⁷⁷ Pub. L. No. 98-620, 98 Stat. 3335 (1984) (codified as amended at 17 U.S.C. §§ 901-914 (1988)). For an excellent analysis of the various aspects of SCPA, *see* Symposium, *The Semiconductor Chip Protection Act of 1984 and Its Lessons*, 70 Minn. L. Rev. 263 (1985). *See also* RICHARD H. STERN, SEMICONDUCTOR CHIP PROTECTION (1986) (providing an in-depth analysis of the SCPA).

⁷⁸ This type of protection is sometimes referred to as *sui generis*, defined as: "Of its own kind or class; *i.e.*, the *only one* of its own kind; peculiar." BLACK'S LAW DICTIONARY 1434 (6th ed. 1990).

⁷⁹ "A semiconductor chip is a fingernail-sized bit of silicon on which an extraordinary number of transistors have been constructed and connected, so that a chip may contain a million or more electronic elements. Most modern computers are comprised of semiconductor chips, among other elements." John A. Kidwell, *Software and Semiconductors: Why Are We Confused?*, 70 MINN. L. REV. 533, 540-41 (1985) (footnotes omitted).

⁸⁰ The specific provision states:

(a) Notwithstanding the provisions of section 905, it is not an infringement of the exclusive rights of the owner of a 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 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

may fully analyze a chip without infringement, and may then use this research to create their own original chips.⁸¹ This stands in stark contrast to the "closed-door" policies of current copyright law.⁸² The overprotection of software, as compared to semiconductor chips, is particularly relevant because of the great similarities between the two technologies, both in the way they are developed and the way they operate.⁸³

C. *Overly Strict Protection as a Hindrance to Software Innovation*

Innovation may be encouraged through the use of two diametrically opposed methods: (1) by freely disseminating information, allowing people to build upon the advancements of others; and (2) by providing incentives for inventors to perform the research necessary for such advancements. When protecting intellectual property, Congress must very carefully balance these two methods in order to best foster innovation in the industry.⁸⁴ The above

in an original mask work which is made to be distributed.

17 U.S.C. § 906(a) (1988).

⁸¹ The exact standard of originality is not precisely defined by the Act. The derivative work must show greater originality than the bare minimum needed for copyrighted works, but need not meet the strict "novelty" standard of patent law. See Leo J. Raskind, *Reverse Engineering, Unfair Competition, and Fair Use*, 70 MINN. L. REV. 385, 399-401 (1985) (analyzing legislative history in order to determine the meaning of "originality"); Stern, *supra* note 49, at 333-36 (discussing the meaning of "original" and "originality" in the Act).

⁸² Congress appeared to rely heavily on the fact that reverse engineering was an established practice of the industry, and was responsible for much of the innovation which had occurred in the development of the field. See Raskind, *supra* note 81, at 385-86; Stern, *supra* note 49, at 327. This rationale, however, applies equally to the software industry, which also has grown incrementally through the study of prior works through reverse engineering. See *infra* notes 114-25 and accompanying text.

⁸³ The dividing line between hardware (such as semiconductor chips) and software is extremely fuzzy. "It is commonly recognized in the computer science community that a customized piece of hardware can be built to do any task that might otherwise be performed by running a program on a general purpose computer." Samuelson, *supra* note 17, at 509.

⁸⁴ See 17 U.S.C. §102(b) (1988) (limiting the extent of copyright protection and prohibiting copyright protection for any "idea, procedure, process, system, method of operation, concept, principle, or discovery"); *Baker v. Selden*, 101 U.S. 99, 107 (1879) (holding that an author's work is not entitled to protection where the idea underlying it and its expression merge). Supporters of the ban against decompilation argue that, if reverse engineering is permitted, other developers will be able to take the most valuable aspects of the software. This, however, results from the fact that the ideas behind a program are often more valuable than the expression of those ideas. See Karjala, *supra* note 7, at 83 (criticizing the idea/expression distinction).

comparisons with other protection schemes illustrate that software protection goes far beyond any balance previously drawn by Congress in the protection of intellectual property. Such protection may stifle innovation, forcing programmers to independently develop basic programming advances that should be freely disseminated.

Overprotecting computer programs is especially dangerous, because software, although classified as a literary work, is a technology.⁸⁵ “[P]rograms remain the technology for *using* computers. They are not designed to communicate information, thought, or feeling to human beings”⁸⁶ Programs are written to control computers—every single line of a computer program is present for a utilitarian reason.⁸⁷ The protection of such technology must be drawn carefully because copyright law is not designed to protect utilitarian works, and as a general policy leaves such works unprotected.⁸⁸ This exclusion of utilitarian works from copyright reflects the particular importance of wide dissemination of technological advancements to the growth of innovation. When drawing a balance for software protection, Congress must be particularly careful to address software as a technology. While overprotecting a novella will likely have little effect upon the field

Although this argument supports the need for a system other than copyright protection for software, it does not in any way justify a breakdown of the traditional idea-expression dichotomy.

⁸⁵ See, e.g., Nimmer & Krauthaus, *supra* note 21, at 20 (“Issues about software protection deal with a creative industry different from print, video or music media. Decisions about protecting software are connected to protecting machine processes and scientific or technological concepts.”).

⁸⁶ Karjala, *supra* note 7, at 38.

⁸⁷ Although other programmers may want to learn by reading programs, the “human reading of a program is a distinctly secondary purpose, deeply subordinate to the primary purpose of causing the computer to perform the intended task.” *Id.*

⁸⁸ Congress explicitly left the utilitarian aspects of an article unprotected through its definition of “pictorial, graphic, and sculptural works”:

Such works shall include works of artistic craftsmanship insofar as their form but not their mechanical or utilitarian aspects are concerned; the design of a useful article, as defined in this section, shall be considered a pictorial, graphic, or sculptural work only if, and only to the extent that, such design incorporates pictorial, graphic, or sculptural features that can be identified separately from, and are capable of existing independently of, the utilitarian aspects of the article.

17 U.S.C. § 101 (1988); see also *Mazer v. Stein*, 347 U.S. 201, 218 (1954). For a well-reasoned argument that utilitarian software should not be copyrightable, see Duncan M. Davidson, *Protecting Computer Software: A Comprehensive Analysis*, 1983 ARIZ. ST. L.J. 611, 678-93 (dissent by Manny Pokotilow).

of literature as a whole, overprotecting a crucial advancement in technology could seriously hamper future innovation in the industry.

Moreover, unlike all other copyrighted works, computer programs require efficiency—one programming idea can be objectively superior to another, and there can be a single “best” way to solve a programming problem.⁸⁹ Limiting the dissemination of a probably superior programming technique will obviously have very harmful effects upon further development in the software industry. Restricting innovation in the software industry is especially troubling because software is used so extensively in the development of technology in other industries. Computers have been essential to recent advances in biotechnology, communications, transportation, manufacturing, and virtually every other field of study. Improperly protecting computer programs will thus have ramifications far beyond the perimeters of the software industry.⁹⁰

The strong protection of computer software has affected all development in the software industry, but most drastically the development of compatible programs. The next Section examines the costs and benefits of compatibility, and concludes that the development of compatible software is crucial to innovation in the computer industry. It argues that, because compatibility is an essential key to software innovation, it is extremely dangerous and counterproductive to discourage such development through current copyright protection. Such a result would defeat the very purpose of copyright: “to promote the Progress of Science and the useful Arts.”⁹¹

III. THE NEED FOR COMPATIBILITY

The inadvertent protection of ideas under copyright may suggest that reverse engineering should be permitted in all instances. There is, however, one case in which reverse engineering is particularly justified—making a new program compatible with existing copyright-

⁸⁹ See Karjala, *supra* note 7, at 39. For example, one programming concept may be faster, or may use memory more efficiently.

⁹⁰ For example, consider the field of three-dimensional graphics software. Restricting advancement in this field would not only affect the software industry, but also would affect the fields of molecular biology, genetics, and chemistry, which rely upon such programs to display complex molecular compounds and model chemical reactions.

⁹¹ U.S. CONST. art. I, § 8, cl. 8.

ed software. As a general notion, compatibility refers to a product's ability to work together easily with another product. As an example of such compatibility, consider a word processing program (WriteIt), and an application which creates graphs (GraphIt). Although each software package may be useful as a self-contained program, the two applications become significantly more productive if made compatible with each other. A user might take figures from WriteIt and send them directly to GraphIt to be automatically graphed. Similarly, a graph created using GraphIt might be sent to WriteIt, where it could be inserted automatically into a research report or a letter. Such compatibility, sometimes referred to as interoperability, would make both WriteIt and GraphIt more useful (and hence more valuable) without diminishing the market for either program.

The compatibility illustrated in this example is often difficult to achieve, even with the mutual cooperation of the programs' authors. At a minimum, a programmer must have a complete specification of the other program's "interface"—a precise description of how the program receives, stores and/or outputs information—in order to make a compatible program.⁹² This information, however, is virtually impossible to determine from the thousands of zeros and ones which comprise the program's object code, the format in which software is generally sold. In most cases, the programmer must decompile the software, changing the object code into source code that humans can understand, and then extracting the interface from source code.

Because reverse engineering⁹³ is often required to achieve interoperability, it is clear that making compatible programs can only be done at the expense of certain copyright protection.⁹⁴

⁹² See Hart, *supra* note 54, at 111-12.

⁹³ See *supra* notes 50-57 and accompanying text (discussing why decompilation is technically a copyright infringement).

⁹⁴ Most courts have been quite unsympathetic to this dilemma, and have held that the need for compatibility does not justify copyright infringement. In *Apple Computer, Inc. v. Franklin Computer Corp.*, 714 F.2d 1240 (3d Cir. 1983), the Third Circuit emphatically stated: "Franklin may wish to achieve total compatibility with independently developed application programs written for the Apple II, but that is a commercial and competitive objective which does not enter into the somewhat metaphysical issue of whether particular ideas and expressions have merged." *Id.* at 1253. In *Lotus Dev. Corp. v. Paperback Software Int'l*, 740 F. Supp. 37 (D. Mass. 1990), the court took an equally negative view of compatibility as an excuse for infringement: "[E]ven if VP-Planner otherwise would have been a commercial failure, and even if no other technological ways of achieving macro and menu compatibility existed, the desire to achieve 'compatibility' or 'standardization' cannot override the rights of authors to a limited monopoly in the expression embodied in their

The resolution of this dilemma can only lie with an examination of the fundamental goal of copyright—the advancement of progress and innovation.⁹⁵ Is current copyright law incorrectly balanced, impeding progress by restricting development of useful compatible software? Would a law allowing reverse engineering to achieve interoperability restrict the progress of computer science by removing incentives to invent? To answer these questions, one must fully examine compatibility, its costs and benefits, and its subsequent effect upon innovation in the software industry.

A. *The Costs and Benefits of Compatibility*

Behind the issue of compatible software is the fundamental question of the merits of standardization. In industries in which large amounts of information must be exchanged,⁹⁶ and in cases in which highly technical products will often be used by relatively

intellectual 'work.'" *Id.* at 69. These decisions indicate that a developer's goal of compatibility with another program should not factor at all into the issues of copyrightability and infringement.

There have, however, been a small minority of cases which are more sympathetic towards the developers of compatible software. In *NEC Corp. v. Intel Corp.*, 10 U.S.P.Q.2d (BNA) 1177 (N.D. Cal. 1989), in which NEC had reverse engineered the microcode for Intel's 8086/88 chip and then independently created a similar set of compatible routines, the district court held that NEC's code was not an infringement because NEC was constrained by its goal of compatibility. The court held that, "[i]n determining an idea's range of expression, constraints are relevant factors to consider In this case, the expression of NEC's microcode was constrained by the use of the macroinstruction set and hardware of the 8086/88." *Id.* at 1188-89 (citations omitted). A favorable view towards compatibility is also present in *E.F. Johnson Co. v. Uniden Corp.*, 623 F. Supp. 1485 (D. Minn. 1985), in which the defendant disassembled the software in the plaintiff's radios and then created compatible radios. The court allowed the copying of certain object code patterns that were required for compatibility with the plaintiff's radios, but found infringement in the duplication of code unnecessary for compatibility. *See id.* at 1502-03.

⁹⁵ *See supra* notes 58-59 and accompanying text.

⁹⁶ One obvious example is that of the telecommunications industry. Until 1984, the domestic telecommunications industry was standardized by AT&T, which would simply dictate its choice for a standard. Due to AT&T's virtual monopoly of the industry, the other "players" would follow this standard to retain compatibility. *See* Joseph Farrell, *Standardization and Intellectual Property*, *LaST Frontier Conference on Copyright Protection of Computer Software*, 30 JURIMETRICS J. 35, 39 (1989); *see also* STANLEY M. BESEN & LELAND L. JOHNSON, *COMPATIBILITY STANDARDS, COMPETITION, AND INNOVATION IN THE BROADCASTING INDUSTRY* (1986) (discussing standardization in the broadcasting industry); Stanley M. Besen & Garth Saloner, *The Economics of Telecommunications Standards*, in *CHANGING THE RULES: TECHNOLOGICAL CHANGE, INTERNATIONAL COMPETITION, AND REGULATION IN COMMUNICATIONS* (Robert W. Crandall & Kenneth Framm eds., 1989) (describing the benefits of standardization in the telecommunication industry).

unsophisticated users,⁹⁷ the creation of standards may be vital to the industry's success. Standardization is used extensively in computer software to "facilitate[] the formation and use of computer networks, the transfer of files among users and across applications, and savings in training costs."⁹⁸

As in other industries, the rewards from compatibility increase as the chosen standard becomes more widely accepted. Such benefits, sometimes referred to as "network externalities,"⁹⁹ are particularly acute in the realm of computer software, in which the rapid development of technology promises to quickly render many programs obsolete. Because users fear being stranded with an unpopular or obsolete program, they will generally buy a program that they believe will be used by many other users. These network externalities tend to have a "snowball" effect—as more and more people use a program, it becomes more valuable, which in turn encourages more people to use it.¹⁰⁰ In many cases, this process will result in a "de facto standard" for the industry, thus giving the owner of the copyright in the "standard" program a complete monopoly over the market.¹⁰¹

In addition to the immediately recognizable benefits, compatible software can indirectly create secondary benefits to the industry and public. For example, both the developer and users of an operating system benefit from the existence of a large pool of compatible software. As more software is written for the operating system, more customers will purchase the system in order to use the growing number of programs available. The programmer profits from such events, especially if her program becomes the de facto standard of the market. Users benefit through the greater choice of software available to them, as well as by the lower prices that

⁹⁷ For example, the benefits of standardization are evident in the case of stereo components. When a consumer purchases a compact disc player, he generally assumes that it will connect to his amplifier without any technical innovation on his part. One would expect that the computer industry, especially that section which caters to non-expert users, would benefit similarly from such compatibility.

⁹⁸ Farrell, *supra* note 96, at 36.

⁹⁹ This name is derived through analogy to telecommunications, "where being 'on the network'—having a telephone—is more valuable the larger the network is." *Id.*

¹⁰⁰ Cf. FREIBERGER & SWAINE, *supra* note 1, at 279 (describing how good publicity and successful initial sales of the IBM PC resulted in third-party development and enormous sales).

¹⁰¹ For a discussion of de facto standards, see *infra* notes 102, 124-29 and accompanying text.

result from increased competition between software developers.¹⁰²

Allowing for compatible software may also spur innovation in the computer industry. Programmers would be able to develop and market a single, technologically superior component, which would be hooked up to other preexisting programs to comprise a complete system. If the programmer instead were forced to independently develop an entire system, the costs of entry into the market might prevent the development of incremental innovations.¹⁰³ Developers may also be prevented from marketing innovations, not by development costs, but by low probability of success due to competitor's de facto monopoly over the market. Allowing the developer to make her innovative software compatible with her competitor's application would make competition more feasible because users would be able to use the new program without losing the many benefits of using an industry standard.¹⁰⁴

¹⁰² Such benefits are dramatically illustrated by the successes of Apple and IBM in popularizing their computer systems. Apple Computer owes much of the success of its Apple II computer to the machine's eight "expansion slots," which allowed third-party manufacturers to create attachments (such as clocks, extra memory boards, modems) which would plug into the machine, as well as its open policy towards third party software developers. See FREIBERGER & SWAINE, *supra* note 1, at 227 ("In fact, it would be hard to overestimate the contribution of third-party developers to Apple."); MORITZ, *supra* note 2, at 233. One of the best examples of the mutual benefits of compatibility is the success of VisiCalc, a program which would perform financial calculations on the Apple II.

Not only did VisiCalc sell, but it helped sell Apples. During its first year, VisiCalc was available only on an Apple disk, and it provided a compelling reason to buy an Apple. In fact, the Apple II and VisiCalc were an impressive symbiotic combination, and it's difficult to say which contributed more to the other. Together they did much to help legitimize both the hardware and the software industries.

FREIBERGER & SWAINE, *supra* note 1, at 230.

Similar benefits accrued to IBM when it first introduced its IBM PC computer with a new operating system (MS-DOS). IBM's decision to encourage third-party development spurred a vast amount of compatible hardware and software. "In turn, the add-on products themselves spurred PC sales, since they increased the utility of the machine." *Id.* at 278. This continuous cycle "served to ratify the PC operating system. MS-DOS quickly became the standard operating system for 16-bit machines." *Id.* at 279.

¹⁰³ See Farrell, *supra* note 96, at 36.

¹⁰⁴ See, e.g., Michael A. Jacobs, *Copyright and Compatibility*, *LaST Frontier Conference on Copyright Protection of Computer Software*, 30 JURIMETRICS J. 91, 100 (1989) ("[T]he benefits that compatibility offers [include] the ability to introduce new products to the market, to innovate, without asking customers to sacrifice their existing investments in data, software and training."). One example of this effect is Borland's "Quattro," an innovative spreadsheet program for personal computers. At the time

Although standardization may have a positive effect upon an industry, it is important to realize that compatibility is a double-edged sword. The costs of creating a standard can be great, especially where society locks itself into a standard which then becomes obsolete. A common example of such an occurrence is the design of the "QWERTY" typewriter keyboard, found today on virtually all English language typewriters and computers. The QWERTY keyboard was originally developed in the nineteenth century as a means of slowing typists down, so that typewriter keys would not stick when two keys were quickly struck in succession.¹⁰⁵ By design, the most frequently used letters are not in the center row, the left hand does most of the work, and common letter sequences require the fingers to make awkward jumps and reaches. The QWERTY keyboard admirably fulfilled its purpose, and became

of its development and release, the spreadsheet market was dominated by Lotus 1-2-3. "Borland felt compatibility was necessary in order to introduce this product to the market; the company advertises that 'Quattro is 100% compatible with The Other Spreadsheet.' The Other Spreadsheet is, of course, Lotus 1-2-3." *Id.*

In Lotus's suit against another 1-2-3 compatible program, Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37 (D. Mass. 1990), the court explicitly rejected this argument, stating:

... Defendants have argued that 1-2-3, and, specifically, 1-2-3's menu structure and macro command facility, has set a de facto industry standard for all electronic spreadsheets. Thus, defendants had no choice, they argue, but to copy these expressive elements from 1-2-3. . . .

... [D]efendant's argument ignores the commercial success of Excel, an innovative spreadsheet that is not compatible with 1-2-3, either in its menu structure or in its macro command facility.

Id. at 78.

This comparison with Excel is improper, however, because Excel is a spreadsheet program designed to run on the Apple Macintosh, while Lotus 1-2-3 is run primarily on IBM computers. When Excel was developed, there was not as great a need to make it compatible with Lotus 1-2-3, because Lotus 1-2-3 was not even available for the Macintosh at the time. See Christopher Lindquist, *Lotus to Unleash Overdue Mac 1-2-3*, COMPUTERWORLD, Dec. 16, 1991, at 8. Furthermore, virtually all Macintosh applications use an interface different from (and more advanced than) the interface of IBM PC applications. To the extent that Excel's interface is different from 1-2-3, it is most likely different out of a desire to be compatible with the standard interface of the Macintosh. See *Microsoft Debuts Rival to "Jazz"*, PC WEEK, May 7, 1985, at 3 ("Excel's design used the common data and screen interfaces of the Macintosh to allow interchange of information from one full-featured program to another without the need for the compromises that are inherent in integrated packages such as Lotus 1-2-3.").

¹⁰⁵ See Farrell, *supra* note 96, at 37; Thomas M.S. Hemnes, *Three Common Fallacies in the User Interface Copyright Debate*, 6 COMPUTER L. & PRAC. 163, 167 (1990). See generally Paul A. David, *Clio and the Economics of QWERTY*, 75 AM. ECON. REV. 332 (1985).

the de facto standard of the industry. Advancements in technology, however, have eliminated the need to slow down typists, and the QWERTY keyboard is therefore technologically obsolete. Modern typewriters generally do not utilize separate hammers that a fast typist may jam, and computers, of course, can keep up with the fastest typist. A number of different and more efficient keyboards have been designed, such as the Dvorak keyboard, but none have been commercially successful.¹⁰⁶ The reason behind the retention of such an obsolete design rests in the substantial durability of standards. Once an industry has been standardized, it can be extremely difficult to break out of that standard, even if it is no longer optimal. Technological leaders who boldly adopt a new design risk losing considerable market share if the industry does not follow.¹⁰⁷ Even if the industry is united in its desire for a new standard, change may still be hampered by disagreement over what that standard should be, how it should be implemented, and when it should take effect.¹⁰⁸

The difficulty of adopting new innovations in place of old standards also results from high transaction costs which can prevent the change from being profitable in the short run. These costs vary depending on the standard and the industry—in the field of computer software, the transactions costs can be remarkably high. For example, a company which decides to switch to a newer, more innovative word-processing program will have to install the program

¹⁰⁶ Although Dvorak originally patented his keyboard, he later dedicated it to the public domain in order to encourage its widespread use. Still, however, QWERTY survives today. See Hemnes, *supra* note 105, at 167 n.58.

¹⁰⁷ This risk is really nothing more than "the Prisoners' Dilemma," applicable to all areas of life where lack of communication and other transaction costs prevent efficient action. In the Prisoners' Dilemma, two criminals commit a crime and are later taken by the police for questioning. Each prisoner is privately told that if he "turns in" his partner, he will be set free. Although the most efficient outcome for the criminals would be for both to remain silent (allowing both to go free), each prisoner will fear being stranded by the other. Thus, the prisoners will likely turn each other in, resulting in jail time for both. See Leo Herzel & Leo Katz, *The Prisoner's Dilemma*, 24 L. SCH. REC. 8, 8-10 (1985).

¹⁰⁸ See Farrell, *supra* note 96, at 37. Professor Farrell draws an analogy to a Western movie, with cowboys riding their horses in the middle of the desert. Before going to sleep, to prevent the horses from running off, they simply tie the horses together. The horses will not all want to leave at the same time, and will not all want to go to the same place. When the cowboys wake in the morning, the horses are merely a few hundred feet away. See *id.*; Joseph Farrell and Garth Saloner, *Competition, Compatibility, and Standards: The Economics of Horses, Penguins, & Lemmings*, in *PRODUCT COMPATIBILITY AS A COMPETITIVE STRATEGY* 1, 10 (H. Lands Gabel ed., 1987).

on each of its computers (and perhaps purchase and install new equipment), train all of its employees in the use of the program,¹⁰⁹ and convert all of its existing documents to the format used by the new program. Even with proper training, productivity will suffer while employees adjust to the new standard. If the software is an integral part of the operation of the business, then more fundamental changes in organization and business practices may be necessary.¹¹⁰ Developers and users are understandably reluctant to adopt such costly changes.¹¹¹

Although the potential restrictions upon innovation are not always evident (or important) to the users of compatible products, standardization may have additional negative effects which will more directly affect software users. Perhaps most significant is a lack of variety in the products available, due to the inevitable design constraints that compatibility imposes. Such a result, however, is by no means assured. Standardization may instead allow for great variety between components of a system, with users able to mix and match different products to customize their system to their individual needs.¹¹²

¹⁰⁹ One consultant has estimated that it costs one thousand dollars to train each user to learn a popular spreadsheet program. See Kathleen K. Wiegner & John Heins, *Can Las Vegas Sue Atlantic City?*, FORBES, Mar. 6, 1989, at 130, 132.

¹¹⁰ Consider, for example, the case of a software company which decides that its products should henceforth be written in the computer language "C" rather than Pascal. Such a change could effectively bring the operation of the entire company to a halt.

¹¹¹ It is worth noting that the reluctance of users to leave a given standard certainly offers developers a number of benefits. For example, if the producer of a line of computers uses a standard operating system for all its machines (or at least makes them "upwardly compatible," so that users can "trade up" to a more powerful computer and still run their old programs), then users will be more likely to purchase several different computers from the line. Consider this explanation of IBM's dominance of the computer mainframe market:

Total software compatibility made it easy for customers to do what IBM wanted them to do, which was to buy several different kinds of 360 computers. A customer could buy a small one now and later on buy a bigger one, or vice versa, without having to re-create any software. Software compatibility strengthened IBM's already tight grip on its customers: they weren't likely to forsake IBM and take their business elsewhere when that meant assuming new expenses and problems with software.

KIDDER, *supra* note 12, at 43.

¹¹² See Carmen Matutes & Pierre Regibeau, "Mix and Match": *Product Compatibility Without Network Externalities*, 19 RAND J. ECON. 221, 222 (1988). For example, users might build a specialized word processing system, separately purchasing a simple text editor, a complex program for advanced searching and replacing, and a program to produce large quantities of form letters. In such a case, the user chooses only the

B. *Why Compatibility is Desirable in the Software Industry*

Having carefully weighed the costs and benefits of compatibility, the question remains: on which side does the balance tip? From the viewpoint of the user, compatibility clearly offers great rewards by making programs easier to use, providing greater productivity, and offering greater networking capabilities.¹¹³ However, the proper analysis must be made in light of the single goal of copyright—the progress of innovation. Simply put, the issue is whether compatibility will increase or decrease innovation in the computer software industry. This Section analyzes the net effect of standardization, and concludes that the benefits of compatibility far outweigh the costs in the computer industry. Furthermore, the small costs of standardization are likely unavoidable, due to the natural tendency to develop *de facto* standards. For these reasons, compatibility is both desirable and necessary in the software industry.

1. Evolution v. Revolution

Interoperability is essential to a particular type of innovation—specifically, the *cumulative development of technology*. By not forcing programmers to “reinvent the wheel,” interoperability allows the concentration of development efforts in the area of improving programs, rather than replicating past achievements. Compatibility allows innovative new products to enter a monopolized market and lowers development costs by allowing a programmer to attach his single innovative component to a preexisting complete system.¹¹⁴

The negative effects of standardization are also concentrated in a particular area of innovation—the *revolutionary idea or design*. The network externalities that make standardization so desirable also make it very difficult to adopt a completely new and revolutionary standard. These benefits, combined with the high costs of learning a new system, make a rapid shift to a new technology unlikely, even if the innovation is a significant improvement over the previous standard.¹¹⁵

Thus, compatibility encourages one type of innovation and discourages another. Which form of innovation is more prevalent

specific components that contain the features she wants.

¹¹³ See *supra* notes 96-100 and accompanying text.

¹¹⁴ See *supra* notes 102-04 and accompanying text.

¹¹⁵ See *supra* notes 105-11 and accompanying text.

in the computer industry? The answer is unmistakably clear—cumulative innovation. Indeed, the rapid early advances in computer science result largely from the willingness of early programmers to completely share all new ideas, developments, and software.¹¹⁶ Software provides a classic example of technological advancement through incremental developments.

The 'broad protection favours innovation' line of reasoning . . . makes a false assumption about the nature of development in the software industry. It assumes that important innovations in software interfaces are revolutionary and not evolutionary in nature. This is wrong. It is impossible to point to a single element of any current mass-market program's interface that did not have a progenitor in one or more prior programs.¹¹⁷

The most innovative programs today, regardless of their specific function, generally owe their origin to a continuous stream of developments by others. An early expression of this notion was made by Sir Isaac Newton, who commented: "If I have seen further it is by standing on ye shoulders of Giants."¹¹⁸ One commentator noted, somewhat more recently, that "[i]f companies are afraid to go to market with what they think are incremental—but distinct—improvements on a basic design, we will become a stagnant industry."¹¹⁹

The user interface of the Apple Macintosh computer is a perfect example of the overwhelming prevalence of cumulative development in the software industry. The interface utilizes a mouse, windows, icons, pull-down menus, and other features in order to make the Macintosh easier to use. Although this system was thought by many to be a bold and revolutionary development,¹²⁰ it is actually a perfect example of the "stepping stone" development prevalent in the computer industry. The user interface of the Macintosh was largely based on advances made by Xerox in its Palo Alto Research

¹¹⁶ For a detailed discussion of the importance of wide dissemination of information and incremental improvements to the success of the computer revolution, see LEVY, *supra* note 44.

¹¹⁷ Hemnes, *supra* note 105, at 167.

¹¹⁸ Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37, 77 (D. Mass. 1990) (quoting a letter from Sir Isaac Newton to Robert Hooke (Feb. 5, 1675/1676)). Because of this statement, the principle of cumulative development is sometimes referred to as the OTSOG principle (On The Shoulders Of Giants). See *id.*

¹¹⁹ Gregg Williams, *A Threat to Future Software*, BYTE, Jan. 1986, at 6.

¹²⁰ For early reviews of the Macintosh, see Thomas Neudecker, *Apple's Macintosh*, INFOWORLD, Mar. 26, 1984, at 82; Gregg Williams, *The Apple Macintosh Computer*, BYTE, Feb. 1984, at 30.

Center (PARC).¹²¹ Steve Jobs, one of the founders of Apple, decided to adopt the technology for use in Apple's new machines after visiting the PARC facility.¹²² "The visits to Xerox became one of those few, crucial events that helped bring some clarity to the shape of Apple's computers."¹²³ Similar experiences can be found throughout the history of the software industry.¹²⁴ The continuing stimulation of such cumulative development, by actively promoting the development of standardization, is the course of action that best promotes the overall growth of innovation in the software industry.¹²⁵

¹²¹ See MORITZ, *supra* note 2, at 298-301.

¹²² See *id.* at 300.

¹²³ *Id.* at 300-01.

¹²⁴ Consider the field of application programs, such as word-processors. The authors of today's leading word-processing programs did not suddenly "invent" great programs. They created their products by incorporating the features of past market leaders, such as MicroPro's "WordStar," itself developed as an improvement to a program called "Electric Pencil." See FREIBERGER & SWAINE, *supra* note 1, at 152-53. In the field of spreadsheet programs, there is no question that the development of successful programs, such as Lotus 1-2-3, was based on previous advances made by software such as VisiCalc. "[Lotus] 1-2-3, like many electronic spreadsheet programs since, could thus be thought of as an evolutionary product that was built upon the shoulders of VisiCalc." *Lotus Dev. Corp. v. Paperback Software Int'l*, 740 F. Supp. 37, 66 (D. Mass. 1990).

¹²⁵ This writer is not alone in this conclusion:

The conferees agree that achieving compatibility between interface programs of these types—that is, programs that serve as hardware-to-software or a software-to-software interface—is a legitimate goal for software competitors. . . . [I]nfringement should not be found where functional compatibility cannot be achieved except through exact or nearly exact copying.

LaST Frontier Conference Report, *supra* note 49, at 22; see also, Farrell, *supra* note 96, at 47 (stating that interfaces should be unprotected so that other software developers are encouraged to achieve standardization); Hart, *supra* note 54, at 114 ("In the author's opinion, it is clearly necessary to provide an exception for obtaining interoperability information on program-to-program interfaces."); Hemnes, *supra* note 105, at 167 (arguing against protection of user interfaces so that compatible products may be developed); Hooten, *supra* note 17, at 102 ("Technological standards may be . . . critical for the commercial success of software. Where would the computer industry be if a standard QWERTY user interface for keyboards did not exist?"); Jacobs, *supra* note 104, at 103-04 (arguing for a system that would allow development of compatible products); Karjala, *supra* note 7, at 58 (finding no unfair advantage in legitimate reverse engineering for the legitimate purpose of compatibility); Nimmer & Krauthaus, *supra* note 21, at 62 (arguing that compatible software and other derivative works should be permitted as long as the use of original work is selective and developmental); Alfred Z. Spector, *Software, Interface, and Implementation*, *LaST Frontier Conference on Copyright Protection of Computer Software*, 30 JURIMETRICS J. 79, 87 (1989) ("I believe it is a technological imperative for the law to encourage system designers to use standardized abstractions."); Richard H. Stern, *Copyright Infringement*

2. The Inevitability of De Facto Standards

It is worth noting that, to the extent that standardization inhibits revolutionary innovation, this negative effect often will be present whether or not third parties are permitted to reverse engineer in order to achieve compatibility. In the software industry, a field characterized by large network externalities, a popular product may quickly gain a de facto monopoly over the market.¹²⁶ This process of "bandwagon standardization," whereby users follow a particular standard precisely because they believe others will follow as well,¹²⁷ generally takes place without any formal agreement or discussion. These standards are arbitrary and involuntary in the sense that no single person or organization actually advocates designating the product as the industry standard.¹²⁸ Instead, they are a reflection of the consumers' desire to buy what everyone else is buying, and their dominance will occur whether compatibility is encouraged or not.

Since standardization will occur regardless of the scope of intellectual property protection, there is no easy way to avoid the built-in costs of having a standard. Discouraging compatibility, through a ban on reverse engineering, would therefore eliminate the innumerable benefits of cumulative development while retaining the costs of standardization (through the bandwagon effect). This result reinforces the conclusion that compatibility is a legitimate goal, both for the computer industry and for copyright law.¹²⁹

by *Add-On Software: Going Beyond Deconstruction of the Mona Lisa Moustache Paradigm and Not Taking Video Game Cases Too Seriously*, 31 JURIMETRICS J. 205 (1991) (arguing that "add-on" programs—compatible software which modifies the function of a preexisting program—should be legal).

¹²⁶ See Karjala, *supra* note 7, at 44-48. "All it takes is widespread public acceptance of one particular manufacturer's product, for whatever reason. Once lock-in begins, it can become self-sustaining." *Id.* at 45-46.

¹²⁷ An excellent example can be found with the IBM PC computer. "IBM did not begin with a large market share, but because users and software-writers thought that IBM was likely to set a standard, its PC did indeed become a standard: a self-fulfilling expectation." Farrell, *supra* note 96, at 40. Many examples of market dominance exist in the field of applications software; for instance, Microsoft Excel holds 90% of the Macintosh spreadsheet market, see Robert Wiggins, *Spreadsheet Skirmishes*, MACUSER, Aug. 1991, at 29, and Lotus 1-2-3, controlled (in 1989) 70% of the spreadsheet market for IBM personal computers, see Richard A. Shaffer, *Is Lotus a Buy?*, FORBES, March 18, 1991, at 128.

¹²⁸ See Farrell, *supra* note 96, at 40.

¹²⁹ The importance of compatibility was recognized in legislation recently passed in the European Community, the result of years of battle over the proper scope of copyright protection for computer software in Europe. The debate was marked by

IV. THE RISKS OF ALLOWING REVERSE ENGINEERING: WILL THE PROBLEM OF PIRACY OFFSET THE BENEFITS OF COMPATIBILITY?

While encouraging compatibility in the software industry will have profoundly positive effects upon innovation, such action can only take place at the expense of a certain amount of copyright protection. In order to determine if copyright law should accommodate compatibility as a goal (by allowing reverse engineering for that purpose), one must carefully examine the negative effects that a decrease in the legal protection of software may produce.

Specifically, one should be seriously concerned with the problem of piracy, defined as the "unauthorized commercial reproduction and sale"¹³⁰ of computer programs.¹³¹ Software pirates can

fierce lobbying by special interest groups composed of the world's leading computer companies. The European Committee for Interoperable Systems (ECIS) was composed of 50 software and hardware manufacturers, such as Fujitsu, Unisys, NCR, and Amdahl. It sought to exclude interfaces from copyright protection and to allow reverse engineering of software, so that developers could develop compatible software. See Keith A. Styracula, *The Adequacy of Copyright Protection for Computer Software in the European Community 1992: A Critical Analysis of the EC's Draft Directive*, 31 JURIMETRICS J. 329, 343 (1991). The Software Action Group for Europe (SAGE), a group vigorously opposed to allowing any form of reverse engineering, was composed of manufacturers such as IBM, Digital, Microsoft, Apple, and Lotus. See *id*; Nicholas Kounoupas, *The European Commission Software Directive, WHICH COMPUTER?*, June 1991, at 134, available in LEXIS, Nexis Library, Currnt File. By all accounts, the battles over reverse engineering and interoperability were the most fiercely fought. See *Problems of the New EC Directive*, FIN. TIMES, May 1991, available in LEXIS, Nexis Library, Currnt File. One observer noted that "[t]he directive has been lobbied ridiculously on the reverse engineering issue." Suzanne Perry, *EC Software Directive Adopted, But Debate Goes On*, REUTERS, May 16, 1991, available in LEXIS, Nexis Library, Currnt File.

The end result of this intense battle was an explicit recognition of the importance of compatibility, and the need to allow reverse engineering to further that goal. See Council Directive 91/250 on the Legal Protection of Computer Programs, 1991 O.J. (L 122/43). The Directive, although banning reverse engineering in general, allows such copying when necessary to achieve "interoperability" between programs. Article 6.1 allows decompilation if required to achieve the interoperability of the independently created program with other programs, provided that:

- the decompilation is performed by the licensee or another person having the right to use a copy of the program, or a person authorized on their behalf;
- the information necessary to achieve interoperability has not previously been published or made available to the person; and
- the acts are confined to the parts of the original program which are necessary to achieve interoperability.

Article 6.2 severely restricts the application of the information obtained through such decompilations, stating that it is not permitted:

- to be used for purposes other than the achievement of interoperability.

destroy the market for a computer program because they can favorably compete with the developer without concern about recouping the costs of research and development.¹³² Software is particularly vulnerable to piracy because of the ease with which it is copied—"[t]he 'pirate' appropriates the technology in a manner similar to that for video tapes, records and cassette tape recordings."¹³³ This ease of duplication drastically affects the market for software by completely eliminating the "head start" advantage that a developer would otherwise retain in the marketplace.¹³⁴ These severe costs have no counterbalancing benefit in terms of further innovation because the pirate contributes nothing new to the software.¹³⁵

Because of the unfair advantages that piracy entails, and the subsequent effects on innovation, scholars, programmers and

-
- to be given to others, except when necessary to achieve interoperability.
 - to be used for any act which would infringe the copyright of the original program, and in particular a program substantially similar in expression to the original.

See *Computer Software Protection Directive Adopted*, Common Mkt. Rep. (CCH), at 1 (May 30, 1991) (describing the contents of the Directive). The Directive correctly recognizes the importance of compatibility as a stimulant to innovation and further recognizes the crucial role that reverse engineering must play in achieving such compatibility. It is essential that the United States follow the lead of the European Community, correcting the imbalance that currently exists in the protection of computer software.

¹³⁰ Nimmer & Krauthaus, *supra* note 21, at 15.

¹³¹ See, e.g., *Vault Corp. v. Quaid Software Ltd.*, 847 F.2d 255, 261 n.13 (5th Cir. 1988) ("In 1983 it was estimated that twenty to thirty percent of the computer software industry's revenues were siphoned off annually by piracy . . ."); William M. Bulkeley, *Software Makers Are Pursuing "Pirates" Around the Globe with Fleets of Lawyers*, WALL ST. J., Dec. 13, 1990, at B1 ("The PC software industry calculates that it loses \$3 billion a year to illegal copies in the U.S., \$5.3 billion a year in Europe and several billion more in other markets.").

¹³² See Nimmer & Krauthaus, *supra* note 21, at 21.

¹³³ *Id.* at 22.

¹³⁴ See, e.g., Karjala, *supra* note 7, at 58 ("[T]here is essentially no lag time between buying a single copy of the desired original program and getting into competition with the original on a large scale."); Nimmer & Krauthaus, *supra* note 21, at 22 ("[T]he ease of copying creates a primary incentive for 'piracy' and . . . reduces 'head start' advantages.").

¹³⁵ To be sure, users will be able to obtain a pirated program at a lower price. However, to the extent that this is a benefit, it will be an extremely short-lived benefit. If writing software is no longer profitable, who will write the consumer's next program? Consider the statement of one computer scientist: "Some computer scientists think all software should be free so that we can easily borrow from each other. They argue that this will promote the greatest innovation. They have a point, but I believe there will be less motivation and funds to move the field forward." Spector, *supra* note 125, at 88.

legislators are united in their belief that piracy should be banned.¹³⁶ Will allowing reverse engineering for the purposes of compatibility suddenly make piracy legal? The answer depends in part on the particular definition of piracy adopted. According to the definition used above—the unauthorized, exact duplication of a product—decompilation will not encourage piracy at all. Pirates do not need to reverse engineer in order to copy; the exact copying of computer programs is a mindless and mechanical task.¹³⁷ Allowing reverse engineering would encourage piracy no more than close analysis of literature would encourage unauthorized photocopying.

Difficulties emerge, however, when the definition of piracy is expanded to include the creation of competing programs which, although not exact copies, appropriate portions of an original program's structure, sequence and organization.¹³⁸ Because elements such as a program's structure are generally accessible only through reverse engineering, there is a concern that a lift of the ban on decompilation would result in their appropriation. This analysis, however, ignores the important distinction between the act of reverse engineering and the subsequent development of a competing or compatible program. Permitting the act of decompilation will allow developers to analyze and study a program, but will not automatically exempt a subsequently developed program from infringement. An author's exclusive right to create copies and derivative works¹³⁹ will still prevent any piracy of her work; if the program is substantially similar to a previous work, then it will still be an infringement of copyright law.¹⁴⁰

Thus, it appears that, at least in theory, lifting the ban on reverse engineering would not upset the current copyright protection against piracy. In reality, however, the tests for infringement are currently too unclear and uncertain to guarantee complete protection. As a result, the ban on reverse engineering is currently

¹³⁶ One commentator has remarked: "Making an exact copy of a program, without analysis or any attempt to understand and improve upon it, should clearly be, and clearly is, a copyright violation. On this point, at least, there is universal agreement." Karjala, *supra* note 7, at 58.

¹³⁷ See *id.* ("[I]t is not necessary to reverse engineer the product, or even to understand the ideas taken, in order to take them.").

¹³⁸ These elements, depending on the particular jurisdiction, may be protected as part of the expression of a program, as compared to its idea. For a discussion of the treatment of this issue in the Third and Fifth Circuits, see *supra* notes 25-35 and accompanying text.

¹³⁹ See 17 U.S.C. § 106(1), (2) (1988).

¹⁴⁰ See *supra* notes 50-57 and accompanying text.

relied upon by the software industry to provide a necessary defense against piracy:

The prohibition against reverse engineering is really being put forward as a substitute for the inquiry whether the software produced from the reverse engineering is a copy of the original software. . . . [T]hat will depend on whether the later software is 'substantially similar' to the original software, which can be a very difficult and uncertain exercise: does one count the number of replicated codes, and if so, what proportion will result in substantial similarity; is it the overall 'look and feel' of the two programs; should similarity of non-literal aspects be taken account of? Prohibiting reverse engineering outright avoids these difficulties by turning the existing law on its head: the act of reverse engineering is the primary copyright offen[s]e, and the close similarity of the two programs is nothing more than circumstantial evidence of the act of reverse engineering having taken place.¹⁴¹

Even accepting this analysis,¹⁴² the concerns about lack of protec-

¹⁴¹ Peter Waters & Peter G. Leonard, *The Lessons of Recent EC and US Developments for Protection of Computer Software under Australian Law*, 13 EUR. INTEL. PROP. REV. 124, 129 (1991) (emphasis added); see also John M. Conley & Robert M. Bryan, *A Unifying Theory for Litigation of Computer Software Copyright Cases*, 63 N.C. L. REV. 563, 597 (1985) ("Substantial similarity, after all, is only circumstantial evidence of copying, to be used when no direct evidence is available."); Susan A. Dunn, Note, *Defining the Scope of Copyright Protection for Computer Software*, 38 STAN. L. REV. 497, 515-20 (1986) (discussing the use of course of development evidence to prove infringement as an alternative to substantial similarity).

¹⁴² A bright-line test for substantial similarity may well be impossible, due both to the fact-specific nature of software infringement cases and to the courts' major difficulties with the idea/expression dichotomy in software. Courts have continually attempted to clarify substantial similarity and idea/expression. See, e.g., *Whelan Assocs. v. Jaslow Dental Lab., Inc.*, 797 F.2d 1222, 1236, 1245 (3d Cir. 1986) (noting that "the line between idea and expression may be drawn with reference to the end sought to be achieved by the work in question" and that, when determining substantial similarity, "the court must make a qualitative, not quantitative, judgement about the character of the work as a whole and the importance of the substantially similar portions of the work"), cert. denied, 479 U.S. 1031 (1987); *Lotus Dev. Corp. v. Paperback Software Int'l*, 740 F. Supp. 37, 59 (D. Mass. 1990) ("If . . . the expression of an idea has elements that go beyond all functional elements of the idea itself, and beyond the obvious, and if there are numerous other ways of expressing the non-copyrightable idea, then those elements of expression, if original and substantial, are copyrightable."). But the tests have been muddled, and academic response has not been glowing. See, e.g., Abramson, *supra* note 17, at 6 (criticizing the court in *Lotus* for treating computer software like any other traditional literary work); J. Dianne Brinson, *Copyrighted Software: Separating the Protected Expression from Unprotected Ideas, A Starting Point*, 29 B.C. L. REV. 803, 832 (1988) (criticizing the *Whelan* court for "confusing the program function with the unprotected idea"); Steven R. Englund, Note, *Idea, Process, or Protected Expression?: Determining the Scope of Copyright Protection of the Structure of Computer Programs*, 88 MICH. L. REV. 866, 908 (1990) (concluding

tion do not justify a complete ban on decompilation. While the problems of proving infringement are substantial, "the solution of totally prohibiting reverse engineering is worse than the difficulties it is designed to cure."¹⁴³

Copyright law should not, by simply applying the label of "piracy," treat a third-party's creation of a derivative computer program the same as the mechanical duplication normally associated with piracy.¹⁴⁴ Third-party developers, unlike pirates, generally cannot undercut the prices of their competition because the independent development of a compatible or similar program is quite expensive. In addition to the costs of reverse engineering, the third-party developer must undertake the major part of the development herself.¹⁴⁵ The creator of compatible software also cannot make a quick entrance into the market, because the processes of reverse engineering and subsequent development are very time consuming.¹⁴⁶ Therefore, the unfair conditions which create the need for a total ban on piracy¹⁴⁷ are not present in the actions of many third-party developers.

Admittedly, the third-party development described above may represent a "best case" scenario. There will certainly be cases in

that a program's structure must be protected under copyright law, "subject to the limitations imposed by the idea-expression and process-expression dichotomies and the merger doctrine"); Peter G. Spivack, Comment, *Does Form Follow Function? The Idea/Expression Dichotomy in Copyright Protection of Computer Software*, 35 UCLA L. REV. 723, 748 (1988) ("The *Whelan* court's rule for dividing idea and expression teeters precariously on the brink between the arbitrary and the ad hoc.") (footnote omitted).

¹⁴³ Waters & Leonard, *supra* note 141, at 129.

¹⁴⁴ "The original developer of innovative technology . . . is likely to label as 'piracy' any use not expressly authorized by her. Simply that individual creators would like more protection in a particular case . . . does not show that we have been wrong all these years in denying more protection to nonpatented innovations." Karjala, *supra* note 7, at 78 n.161.

¹⁴⁵ See Jacobs, *supra* note 104, at 102. "It is generally agreed that testing a product and debugging it comprises about 50 percent of the development effort. Detailed design and coding is about 35 percent of the development effort. Functional design, the specification of the functions and the interfaces, represents 15 percent of the development effort." *Id.* Therefore, in a case where a third-party independently creates a compatible piece of software, he saves at most 15 percent of his development costs. There would not appear to be a "free rider" problem in such a case. See Karjala, *supra* note 7, at 58 ("[R]everse engineering . . . provides the competitor in the usual case with little if any advantage in reduced development costs.").

¹⁴⁶ See, e.g., Nimmer & Krauthaus, *supra* note 21, at 22 ("If the second developer must adapt and invest resources to learn and apply the technology, the cost advantages are reduced; the activities require time. This retains commercial advantage for the original developer.").

¹⁴⁷ See *supra* notes 130-34 and accompanying text.

which third-party developers, under the guise of compatibility, make no more than cosmetic changes to a pre-existing program.¹⁴⁸ The possibility of such an occurrence, however, calls for a method of separating the good development activity from the bad, rather than an indiscriminate prohibition on all activity. The solution proposed in this Comment works within the protection scheme of copyright, providing a "compatibility exemption" for reverse engineering activities that meet certain criteria.

V. A PROPOSED SOLUTION: THE COMPATIBILITY EXEMPTION

There are many activities which, although technically violations of copyright law, are permitted due to their socially useful nature. These activities are generally protected under the doctrine of fair use, set forth in § 107 of the Copyright Act:

Notwithstanding the provisions of section 106, the fair use of a copyrighted work, including such use by reproduction in copies or phonorecords or by any other means specified by that section, for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research, is not an infringement of copyright. In determining whether the use made of a work in any particular case is a fair use the factors to be considered shall include—

- (1) the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;
- (2) the nature of the copyrighted work;
- (3) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
- (4) the effect of the use upon the potential market for or value of the copyrighted work.¹⁴⁹

The classification of reverse engineering as a fair use makes sense: although decompiling software involves unauthorized copying, it should be allowed in order to achieve compatibility because of its positive effect upon innovation.

¹⁴⁸ See, e.g., *Apple Computer, Inc. v. Franklin Computer Corp.*, 714 F.2d 1240, 1245 (3d Cir. 1983) (noting that the "variations that did exist were minor, consisting merely of such things as deletion of reference to Apple or its copyright notice"), *cert. dismissed*, 464 U.S. 1033 (1984).

¹⁴⁹ 17 U.S.C. § 107 (1988). For a very complete description of the fair use exemption in copyright law, see WILLIAM F. PATRY, *THE FAIR USE PRIVILEGE IN COPYRIGHT LAW* (1985).

Nonetheless, it is clear that most reverse engineering fails each of the "four factors" which Congress has established to test for fair use.

A quick examination of typical reverse engineering under the statutory four fair use factors also indicates that fair use is not available. The nature and purpose of the use is entirely commercial; the copyrighted source code is an unpublished work subject to a narrow scope of fair use; the entire work is copied; and since the use is commercial, potential harm to the market for the original is presumed.¹⁵⁰

Indeed, no court has ever held any act of decompilation to be a fair use.¹⁵¹ The unavailability of the fair use exemption is largely due to the heavy emphasis that the fair use doctrine places on the financial reward of the original author. A use qualifying for this exemption must not only be socially useful, but must also avoid interfering with the author's ability to profit from her work. Reverse engineering, although a net benefit to society, clearly does not fall into this category—the creation of compatible or competing software will obviously affect the author's ability to capitalize on her work.

The heavy emphasis that fair use analysis places upon non-profit, non-commercial use precludes its use as a solution to the reverse engineering problem.¹⁵² The need therefore exists for a separate

¹⁵⁰ PATRY, *supra* note 149, at 401 (footnote omitted). Some might argue that the test has not been applied fairly to reverse engineering. For example, consider the nature of the copyrighted work. Courts have been extremely protective of unpublished works, and have steadfastly refused to find fair the use of an unpublished work. See *Harper & Row, Publishers, Inc. v. Nation Enters.*, 471 U.S. 539, 555-60 (1985) (finding one magazine's "scoop" of prepublication excerpts of President Ford's memoirs regarding his pardon of President Nixon not a fair use); *Salinger v. Random House, Inc.*, 811 F.2d 90, 98-99 (2d Cir.) (holding issue of injunction permissible where biographer had taken substantial portions from famous author's copyrighted unpublished letters), *cert. denied*, 484 U.S. 890 (1987). This protection stems from "the author's right to control the first public appearance of his expression." *Harper & Row*, 471 U.S. at 564. In the case of software, however, a program can remain unpublished even after it has been disseminated to the public, due to the unreadable nature of its object code. Therefore, when considering whether reverse engineering is a fair use, one should recognize that the original author has already exploited his "unpublished" work.

¹⁵¹ See Hart, *supra* note 54, at 113.

¹⁵² Consider the statement of one industry representative, made before Congress: "[T]he twin goals of certainty and encouragement of innovation can be achieved only if legitimate reverse engineering is permitted. We feel that existing 'fair use' provisions of Section 107 of the Copyright Law may not be sufficient, however, as they tend to emphasize non-commercial purposes." *Copyright Protection for*

“compatibility exemption” that, instead of distinguishing between non-profit and commercial uses, will distinguish between legitimate software development and unwanted piracy. Legitimate development would be determined with respect to the fundamental goal of copyright—the growth of innovation in the software industry. In developing this exemption, the framework of the current test of fair use is a good starting point. Each of the factors, however, should be analyzed with the goal of distinguishing piracy from legitimate development efforts.

The first three factors for determining fair use—the purpose and character of the use, the nature of the copyrighted work, and the amount and substantiality of the portion used—are essential in determining whether a given decompilation deserves exemption. The compatibility exemption cannot, however, include the last factor—the effect on the original program’s market or value. The consideration of this factor would disqualify all reverse engineering from the exemption, because all compatible software will have some detrimental effect upon the original author’s ability to profit from her work, or at least upon the author’s ability to market derivative works. The result would be a nullification of the entire exemption. The compatibility exemption therefore uses three factors in its determination of which uses should be permitted. This portion of the Comment analyzes each of these factors and specifically applies them to situations in which a program is reverse engineered for the purpose of achieving compatibility.

A. *Purpose and Character of the Use*

In traditional fair use analysis, the chief issue in considering the purpose and character of the use was whether or not it was commercial. It is widely acknowledged that most reverse engineers intend to create and market a new product.¹⁵³ As noted above, however, one must not ask which instances of reverse engineering are acceptable to the first author, but rather which instances will fulfill the copyright law’s goal of increasing innovation. To this end,

Semiconductor Chips: Hearings on H.R. 1028 Before the Subcomm. on Courts, Civil Liberties, and the Administration of Justice of the House Comm. on the Judiciary, 98th Cong., 1st Sess. 201 (1983) (statement of NEC Electronics U.S.A., Inc.).

¹⁵³ See, e.g., PATRY, *supra* note 149, at 400 (acknowledging that the typical reverse engineer does not decompile a program “for the greater public welfare but [does so] only for the commercially selfish purpose of increasing . . . profits by marketing a product in direct competition with the original”).

the relevant inquiry concerns the *nature of the compatibility sought*. Many of the legal problems of compatible software stem from the misconceived notion that there is only one form of compatibility. In fact, there are a number of ways in which a program may be compatible with another, and an understanding of these variations provides an excellent framework for determining when compatibility is desirable.

Software is "vertically compatible" when it does not perform the same function as an underlying program, but instead connects to it in some way. Thus, two programs are vertically compatible when the output of one may be sent to the input of the other. Application programs provide examples of this form of compatibility, since sending information from one program to another greatly increases a user's productivity.¹⁵⁴ Vertical compatibility is also present where one program is able to "run" the other, or where one program can "ask" another program to perform a predetermined function. The operating system and application programs of Apple's Macintosh computer exemplify this point. The application program generally does not draw a rectangle, check to see if the mouse was moved, or perform many other functions by itself. Instead, the application will "ask"¹⁵⁵ the operating system to perform the desired function and return to the application when it is finished. This form of compatibility is essential because an application program simply will not work if it is not compatible with the computer's operating system.

Vertically compatible programs share a number of important characteristics which make their development especially desirable. First, because a vertically compatible program does not perform the same function as the original program, there is no problem of piracy. Vertically compatible programs, by definition, cannot copy the entire structure or organization of the original program, let alone the exact source or object code.¹⁵⁶ Second, vertically

¹⁵⁴ Consider the example of the Writelt and GraphIt programs described earlier. See *supra* text preceding note 92.

¹⁵⁵ Although the inner workings of computers are more easily understood if they are humanized, it should be clear that programs do not actually ask, talk, or understand. Communication between operating systems and application programs is generally performed via a series of routines known as "system calls" or "a toolbox." See, e.g., APPLE COMPUTER, INC., 1 INSIDE MACINTOSH (1985) (describing the toolbox of the Apple Macintosh). For example, an operating system is designed so that it will draw a rectangle whenever it receives the instruction "DrawRectangle." The application program will then be written so that it sends the "DrawRectangle" instruction whenever a rectangle must be drawn on the screen.

¹⁵⁶ The possibility exists, however, that some amount of copying is necessary to

compatible programs tend to increase the market for the original product, because they rely upon the existence of the original program for their effective operation. Finally, vertically compatible programs are original and innovative works. In order to create a vertically compatible work, a third-party developer must contribute time, money, and creative thought in the same manner as the original program's author. The lines of code that achieve compatibility are often minimal in comparison to the whole compatible work. Furthermore, the third-party developer incurs the often considerable costs of reverse engineering.¹⁵⁷ For all of these reasons, reverse engineering should be permitted if necessary to achieve vertical compatibility.¹⁵⁸

On the other hand, programs are "horizontally compatible" if they perform essentially the same function. Horizontally compatible programs will, if given the same input, produce the exact same output. Such compatibility generally will occur between application programs,¹⁵⁹ often because the original program has created a de

achieve such vertical compatibility. This problem is discussed further in light of the third factor of fair use—amount and substantiality of the portion used. *See infra* notes 174-78 and accompanying text; *see also* *Vault Corp. v. Quaid Software Ltd.*, 847 F.2d 255, 267-68 (5th Cir. 1988) (permitting the copying of 30 characters from the original work in order to make it vertically compatible).

¹⁵⁷ *See supra* notes 144-47 and accompanying text.

¹⁵⁸ There is one subcategory of vertically compatible programs that may not be desirable—those programs that destroy or counteract the effectiveness of the original program. For example, in *Hubco Data Prods Corp. v. Management Assistance Inc.*, 219 U.S.P.Q. (BNA) 450 (D. Idaho 1983), the plaintiff placed "governors" which restricted the power of his operating system on certain versions of his program which then were sold at a lower price. The defendant developed and sold a program that would remove the "governors," thereby restoring full power to the plaintiff's operating systems. To do this, the defendant's program would compare line-by-line a "full power" operating system (stored in its memory) with the user's "less powerful" operating system, removing any differences from the latter. The court held that the method used by the defendant's program could be a copyright infringement, and therefore granted a preliminary injunction to the plaintiff. *See id.* at 452, 457-58.

A more common example of this genre of "destructive programs" are the "copying" programs. Manufacturers often use "copy protection" software to physically prevent the unauthorized duplication of their programs. Copying programs, through a variety of methods, undo this protection, thus allowing users to freely pirate the manufacturer's program. This type of compatibility, however, has been explicitly upheld by the Fifth Circuit. In *Vault Corp.*, the court allowed the defendant's copying of 30 characters by the "copying" program (RAMKEY), noting that "Vault's program and RAMKEY serve opposing functions; while Vault's program is designed to prevent the duplication of its customers' programs, RAMKEY is designed to facilitate the creation of copies of Vault's customers' programs." *Vault Corp.*, 847 F.2d at 268.

¹⁵⁹ This result occurs because other forms of software, such as operating systems

facto standard in the industry.¹⁶⁰ In such a case, developers must make their programs "act" like the de facto standard program, or risk being shut out of the marketplace. Pirates often justify their production of near-exact copies of a program by claiming the need for horizontal compatibility. One should not, however, assume that all horizontally compatible programs are exact copies or essentially identical replicas of the original. Rather, such programs may contain vast improvements over the original and, if independently written, may have completely different object code, source code, structure, and organization.¹⁶¹ The relevant inquiry should be whether the compatible program exhibits sufficient originality and innovation,¹⁶² through the addition of substantial improvements, to justify its development through reverse engineering. A decision

and microcode, are generally sold in conjunction with the computer itself. Because every computer owner will already own a copy of the original operating system and microcode, there is not much of a market for programs that will perform the same tasks. Application programs, however, are generally sold separately, thus creating a market for horizontally compatible programs.

¹⁶⁰ See *supra* notes 126-127 and accompanying text.

¹⁶¹ An excellent example of this is the Borland "Quattro Pro" spreadsheet program, compatible with the Lotus 1-2-3 spreadsheet, which critics consider to be superior in many ways. See Jacobs, *supra* note 104, at 100; John T. Soma et al., *A Proposed Legal Advisor's Roadmap for Software Developers: On the Shoulders of Giants May No Breachers of Economic Relationships nor Slavish Copiers Stand*, 68 DENV. U. L. REV. 191, 216 (1991):

There is no question, overall, that Quattro Pro's user interface is substantially different from the Lotus 1-2-3 user interface, except in one respect. The Borland product can be executed in a "Lotus 1-2-3 emulation mode," that is, by specifying a particular command the Borland spreadsheet incorporates the Lotus 1-2-3 menu structure into its own.

Id. This feature was included because "[Lotus] 1-2-3 is the de facto standard for spreadsheets, [thus] any company hoping to achieve significant success must be compatible with it." Peter H. Lewis, *When Computing Power is Generated by the Lawyers*, N.Y. TIMES, July 22, 1990, at F4.

Despite these justifications, Lotus Development Corporation, encouraged by its victory in *Lotus Dev. Corp. v. Paperback Software Int'l*, 740 F. Supp. 37 (D. Mass. 1990), has instigated a currently pending infringement suit against Borland International. See Carissa Casey, *Lotus v. Borland Finds New Twist: Lotus Development Corp., Borland International Look-and-Feel Law Suit*, PC USER, Oct. 23, 1991, at 18, 18, available in LEXIS, Nexis Library, Currnt File (describing the suit between Lotus and Borland).

¹⁶² Note that this "innovation" requirement is not the same as the originality requirement that determines whether or not a work is copyrightable. While a low standard of originality is used to determine copyrightability, a significantly higher standard should be used when considering whether an infringing use should be exempted. The originality standard of the SCPA would likely strike a good balance. See *supra* notes 80-81 and accompanying text.

will depend largely upon the third factor of the compatibility exemption—the amount and substantiality of the portion used.¹⁶³

B. *The Nature of the Copyrighted Work*

When weighing the nature of the copyrighted computer program, one should not consider, as is the case in traditional fair use analysis, whether or not the program is published. The source code of a program remains “unpublished” long after the program’s wide-spread sale to the public, since software can perform its utilitarian function while in its unreadable object code format.¹⁶⁴ The fierce protection of unpublished works stems from the strong desire to allow the author to control the first publication of her work¹⁶⁵—a policy which is satisfied in the case of computer software.

The inquiry under this factor should not concern whether the original program has been published, but should instead ask to what extent banning the decompilation of the original program would have a detrimental effect upon innovation. If the copyrighted computer program has become a *de facto* standard, then reverse engineering must be allowed in order to prevent a single company from practically excluding all further innovative software from the market.¹⁶⁶ The existence of a *de facto* standard is admittedly an uncertain and fact-specific determination. Most cases, however, will likely fall far to one side of the blurry dividing line, simply because developers will reap the greatest benefits by making their programs compatible with those most popular in the industry.

When assessing the nature of the copyrighted program, the specific “level” of the software—where it fits between the hierarchy from hardware to user—is also an important consideration. To understand the concept of a program’s “level,” one must ask why software is useful at all, given the fact that a customized piece of hardware can implement any task that a piece of software can perform.¹⁶⁷ Computer programs are useful because they “make it possible for one machine—a computer—to be many machines.

¹⁶³ See *infra* notes 174-78 and accompanying text.

¹⁶⁴ This is an advantage to developers who, by keeping programs unpublished, are able to secure trade secret protection in addition to copyright protection. See *supra* notes 62-66 and accompanying text.

¹⁶⁵ See *supra* note 150.

¹⁶⁶ See *supra* notes 126-29 and accompanying text.

¹⁶⁷ See, e.g., Samuelson, *supra* note 17, at 510.

Programs, in effect, tell the computer what kind of machine to be."¹⁶⁸ This function is not performed by one single program, but is instead performed by a hierarchy of programs which, when layered upon each other, successively narrow the computer's task to a specific function.¹⁶⁹ As a result, every computer program must be compatible with the "layer of software" beneath it in order to work.

The "level" of a program within the software hierarchy is therefore important, because allowing a de facto monopoly over a program near the bottom of this hierarchy (such as the micro-code¹⁷⁰ or an operating system¹⁷¹) could stifle development and innovation of "higher level" software on the computer. The computer program Microsoft Windows 3.0 exemplifies this point; this operating system for IBM computers is quickly becoming a standard.¹⁷² If Microsoft were to withhold the description of the Windows interface from third-party developers, it could control all future development of all application programs for that particular market.¹⁷³ The lower the "level" of the software, the more likely its decompilation should be an exempted use.

Thus, when considering the nature of the copyrighted work, one should consider the characteristics of the program that may have an inordinately negative effect upon innovation in the industry. These include, but are not limited to, the level of the copyrighted program and the status of the program as a de facto standard.

¹⁶⁸ *Id.* (footnote omitted).

¹⁶⁹ See Samuelson, *supra* note 10, at 680 ("[W]e must realize that it is not the application program alone that performs the task Nor is it the hardware alone. Rather, it is the complex hierarchy of programs and hardware that, while interacting with one another, works as a unit to perform a particular application task."). For an example of this interaction between programs, see *supra* note 155.

¹⁷⁰ See *supra* note 13 and accompanying text.

¹⁷¹ An operating system provides a "platform" from which a user can choose and run specific programs. Operating systems can also include "a toolbox," a set of common programs and routines that may be accessed and run by any application program. See *supra* note 12 and accompanying text.

¹⁷² See *Microsoft to Launch Windows Ads on TV*, REUTERS, Jan. 23, 1992, available in LEXIS, Nexis Library, Currnt File (noting that Microsoft has already reached about 20 million of 50 million desktop machines capable of using the program).

¹⁷³ In practice, this is probably not likely because an operating system generally will not become popular unless there are a large number of programs written for it. Thus, developers generally distribute the interface information for their operating systems. But note that Microsoft is currently under investigation by the FTC for antitrust violations. See Brit Hume, *Of Mice and Software: The Antitrust Case Against Microsoft*, WASH. POST, Apr. 22, 1991, Washington Business, at 17.

C. *The Amount and Substantiality of the Portion Used*

The third and final factor addresses the material actually taken from the copyrighted work. Fair use analysis considers "the amount and substantiality of the portion used in relation to the copyrighted work as a whole."¹⁷⁴ This test, however, incorrectly focuses on simply preserving the author's monopoly—exempting only the use of small and inconsequential fragments of the copyrighted program that would not be harmful to the inventor's financial reward. The goal of this test should instead be to separate the desirable instances of decompilation from the undesirable ones. To this end, the proper consideration should be the amount and substantiality of the portion used *in relation to the compatible work as a whole*. Such a test would properly focus upon the amount of originality and innovation embodied in the derivative work, permitting the development of new and innovative products while preventing the proliferation of unoriginal "knock-off" programs.

This analysis would apply equally well to both vertically and horizontally compatible software. Taking only the interface specifications of a program would represent the bare minimum of copying because a new program can achieve compatibility only by matching its interface to that of the original program. If a compatible program used only the bare interface specifications of the original, then the remainder of the work would be original and reverse engineering would therefore be exempted.¹⁷⁵ On the other hand, if a developer took large portions of the original work, she would have to make substantial innovations in order for her decompilation to be exempt.¹⁷⁶ Reverse engineering efforts would, in effect, depend upon the resultant contribution to innovation in the industry, rather than the resultant interference with the original author's market. This method would be more efficient in practice because it avoids many of the difficulties in

¹⁷⁴ 17 U.S.C. § 107(3) (1988).

¹⁷⁵ By definition, most vertically compatible software would fit into this category. The application of this third factor therefore meshes well with the first factor, which would exempt all vertically compatible works.

¹⁷⁶ This would decide the two Lotus cases in a satisfactory manner. Both Borland and Paperback Software took large portions of Lotus 1-2-3's user interface. See *Lotus Dev. Corp. v. Paperback Software Int'l*, 740 F.Supp. 37, 68-70 (D. Mass. 1990); Casey, *supra* note 161, at 18. Borland, whose Quattro Pro program is substantially original and innovative, would be exempted, while Paperback Software, whose VP-Planner was more or less an unoriginal clone, would not be exempted.

distinguishing idea and expression in computer software.¹⁷⁷ If a compatible program used the basic structure of the original program, the original program's decompilation would be exempt if the compatible program was substantially original, regardless of whether the copied structure constituted idea or expression.¹⁷⁸

D. *A Summary of the Compatibility Exemption*

This Comment proposes a separate exemption for decompilation required to achieve compatibility with existing software. The exemption applies three factors to distinguish legitimate development of compatible works from piracy. A use would qualify for the exemption only if the information needed for compatibility was not already made available to the third-party developer. Such a compatibility exemption might read as follows:

Notwithstanding the provisions of sections 106, the analysis, study, decompilation, or disassembly of a copyrighted computer program for the purpose of creating a legitimate compatible work is not an infringement of copyright. In determining whether a compatible work in any particular case is legitimate, the factors to be considered shall include—

- (1) the purpose and character of the compatible work, including whether such work is vertically or horizontally compatible;
- (2) the nature of the copyrighted computer program; and
- (3) the amount and substantiality of the portion of the copyrighted computer program used, in relation to the compatible program as a whole.

The terms of this section shall not apply if the information necessary to achieve compatibility has been made available by the author of the copyrighted work.

This compatibility exemption would efficiently distinguish between desirable and undesirable development of compatible software. By explicitly recognizing the importance of compatibility, it would spur the cumulative innovation that is so essential to the software industry. In addition, the compatibility exemption will inject

¹⁷⁷ See *supra* notes 25-35, 142 and accompanying text.

¹⁷⁸ Of course, there will be instances in which a program copies the structure of another program without adding anything original. In such a case, the issue of whether the structure was idea or expression will undoubtedly arise. This, however, is really more a question of infringement rather than exemption. The compatibility exemption focuses not upon whether there was an infringement, but whether the particular effort to achieve compatibility was desirable or not.

predictability into an otherwise uncertain field. Although the exemption is by no means a bright-line test, it clearly exempts the development of vertically compatible works and the legitimate development of innovative horizontally compatible works. Moreover, it gives weight to the presence of a *de facto* standard in the industry and prevents manufacturers from tying up important "lower level" technology that will stifle innovation. Finally, it avoids much of the uncertainty brought to copyright law by the "substantial similarity" and "idea-expression" tests, which paralyze development in the software industry.

CONCLUSION

The development of the computer software industry has been characterized by cumulative innovation, a process whereby developers study existing products, make improvements, and subsequently develop competing products. The production of compatible software is an integral part of such development because it allows innovators to improve upon already successful works and facilitates entrance into markets dominated by *de facto* standards. The decision to protect software under copyright law, however, has jeopardized much of this innovation by eliminating developers' ability to study existing software through decompilation. Although the case law in this area is not in any way conclusive, there is a legitimate possibility that the strong protection currently afforded to computer software will eventually stifle development in the industry.

The current overprotection of software could largely be alleviated by permitting the decompilation of programs, thus allowing third-party developers to extract the ideas and create compatible software. The chief stumbling block in the efforts to enact such reform has been the specter of piracy, in which third parties mindlessly copy existing software without incurring the burden of development costs, or contributing any thought, analysis, or innovation to the industry. Although tests for copyright infringement are designed to prevent such piracy, in reality, these tests have been extremely uncertain and unpredictable. As a result, the industry has instead focused on the technical infringements that occur when a program is decompiled, eliminating all legitimate third party development in the process. A total prohibition on reverse engineering, however, is an extremely costly solution to the problem of piracy.

The compatibility exemption described in this Comment will properly balance the rights of original authors and subsequent third party authors, with the twin goals of preventing piracy and encouraging innovation in the software industry. This exemption distinguishes between the legitimate cumulative development that is so vital to the industry, and the piracy which continually threatens to destroy it. The compatibility exemption allows desirable reverse engineering efforts and offers a degree of predictability that is sorely needed in this area of the law. Perhaps most importantly, the emphasis of the law will finally center on encouraging the development of innovative new software—a goal which serves the ultimate purpose of copyright law.