January 1985

Trade Secret Law: An Impediment to Trade in Computer Software

Jay R. Dratler

Follow this and additional works at: http://digitalcommons.law.scu.edu/chtlj

Part of the Law Commons

Recommended Citation
Available at: http://digitalcommons.law.scu.edu/chtlj/vol1/iss1/1

This Article is brought to you for free and open access by the Journals at Santa Clara Law Digital Commons. It has been accepted for inclusion in Santa Clara High Technology Law Journal by an authorized administrator of Santa Clara Law Digital Commons. For more information, please contact sculawlibrarian@gmail.com.
TRADE SECRET LAW: AN IMPEDIMENT TO TRADE IN COMPUTER SOFTWARE

Jay Dratler, Jr.*

I. INTRODUCTION

Computer software is a major item of trade today in the United States and throughout the world. Estimates in 1984 of the value of the worldwide market for computer software range from $10 billion to $21 billion.¹ This figure does not include the value of software incorporated into equipment and machinery — an increasingly common practice. Estimates of future software commerce indicate a potential for enormous growth in both the dollar value and commercial importance of software.² Despite the growing commercial importance of computer software, the American legal system has not evolved an effective and efficient means of protecting and facilitating software commerce. For reasons described more fully below, the three traditional pillars of intellectual property law — patents, copyrights and trade secrets — are all peculiarly inappropriate for protecting computer software and encouraging investment in research and development without impeding everyday business and the development of new channels of distribution.

Patents are difficult to obtain, expensive and time consuming. Moreover, they do not protect some of the most important aspects of software. Copyrights provide good protection for the expression


². See Bus. Wk., Feb. 27, 1984, at 75 (estimating $35 billion market by 1989); N.Y. Times (Sunday ed.), Mar. 27, 1983, § 12, at 22; Shakeout, supra note 1 at 238 ($50 billion by 1988).
of software in specific code in a particular computer language, but
they provide uncertain, if any, protection for the underlying ideas
and techniques. Trade secret law, which in theory can protect
ideas, techniques and algorithms that are not generally known, in
practice requires cumbersome and uncertain precautions for protec-
tion at each level of each distribution channel. The result of the
inability of the existing legal framework to accommodate software
comfortably is unjustified legal complexity and excessive transac-
tion costs, even in pedestrian business transactions. The state of the
law at present is such that almost every transaction requires a writ-
ten agreement drafted or negotiated by attorneys.

This article analyzes the state of the law protecting intellectual
property rights in computer software, discusses current business
practices used in attempting to deal with that body of law, and pro-
poses legislative solutions to facilitate commerce in software.
Although there are aspects of copyright law that impede fluid com-
merce in software, notably the first-sale doctrine discussed below,
this article focuses primarily on the problems associated with trade
secret law.

The second section of this article gives an overview of protec-
tion of computer software by patents, copyrights and trade secret
law. The third section discusses the significant gap in protection of
software between patents and copyrights. It then discusses how
trade secret law, albeit imperfectly, fills that gap. The fourth sec-
tion discusses the traditional requirements for trade secret protec-
tion, their applicability to computer software and their
inappropriateness to the software business as currently conducted.
The final section proposes legislative solutions to the problem of a
legal system that seems not to fit the needs of the software industry.

II. OVERVIEW OF LEGAL PROTECTION OF SOFTWARE

A. What is Software?

The most basic form of software is known as “binary code” or
“object code.” In this form, the computer instructions are repre-

3. The word “software” derives from the contrast between the intangible computer
program instructions and the tangible “hardware,” that is, the physical computer and its
peripheral devices. To the author’s knowledge, there is no standard legal or technical defini-
tion of “software,” and its usage in practice depends upon context, custom and definitions in
particular written agreements. Sometimes the term includes paper documentation, and some-
times it does not.

4. For recent discussions of legal protection for software, see, e.g., Davidson, Protect-
ing Computer Software: A Comprehensive Analysis, 23 Jurimetrics J. 339, 340-48 (1983);
presented as binary numbers, that is, a series of ones and zeros. This is the form in which the instructions are implemented in the computer, the "ones" representing a closed switch or "on" state, and the "zeros" representing an "open" switch or "off" state. Binary code can be represented in both human-readable and machine-readable form. When printed in human-readable form, it is usually translated into a more compact number system, such as octal, hexadecimal, or (less commonly) the decimal system used for ordinary arithmetic.

An important aspect of binary or object code is that, even in human-readable form, it is intelligible to people only with great difficulty. In binary form, software consists only of a long string of numbers, some of which represent computer instructions and some of which represent the locations within the computer's memory where data or computer instructions are stored. With no advance knowledge of the software's nature or purpose, it may be difficult, if not impossible, to decide which numbers represent instructions and which represent memory locations, let alone to determine the grand design of all the minute machine operations that the instructions represent. While there are indications in the computer industry that computer programs have been designed to perform this herculean task, those programs by their nature would be specific to a particular computer language and operating system, if not to a part-

5. "Binary" numbers are numbers to the base two. In this number system, each digit to the left of the decimal point differs from its neighbor to its right by representing higher power of two. For example, the binary number 1101 represents one times 2\(^0\) (that is, 1) plus zero times 2\(^1\) plus one times 2\(^2\) plus one times 2\(^3\), for a total of 13 in decimal notation.

6. Octal numbers are numbers to the base eight, and hexadecimal are numbers to the base sixteen. In conventional hexadecimal notation, single digits for the numbers ten through fifteen are represented by the capital letters A through F, respectively. For example, the decimal number 1984 is expressed in binary notation as 11111000000, in octal notation as 3700 and in hexadecimal notation as 7C0. Since the octal and hexadecimal number systems are based upon powers of two, they have the advantage of easy conversion to and from binary numbers, but they are more compact when printed than binary numbers, as the foregoing examples show.

7. When printed, binary code consists of a huge block of ones and zeroes. In order to interpret binary code, this block would have to be broken into individual numbers corresponding to the instructions and memory locations that the numbers represent and then converted into symbols that people can read, such as separated decimal numbers (or, for trained programmers, octal or hexadecimal numbers) for the memory locations and letter-coded mnemonics for the instructions that the binary codes represent. This process can be done by hand, but it is sufficiently tedious to be prohibitively inefficient for all but the shortest programs.

8. In allegations that certain IBM-compatible computer manufacturers had copied International Business Machines Corporation's proprietary operating systems, IBM has made very specific claims regarding the precise percentages of material that was copied. These percentages could have been derived from statistical analysis, but may also have been
ticular computer. In the general case, the use or conversion of object code into a form that can be used by people for some purpose other than execution in a computer system is a formidable task.

Besides object code, the other major form of software is "source code." Source code is the original set of instructions for the computer, written not in the binary language of computers, but in a language designed to be used by people for computer programming. There are many such "higher-level languages." To the lay person, statements in these languages look like stylized and abbreviated English, punctuated by mathematical symbols. To computer programmers, these languages represent a means to write computer instructions in abbreviated and highly efficient form, subject to precise rules of structure, usage and syntax. Software in source code form is intended to be, and in fact is, intelligible to trained programmers. Unlike object code, source code can be used to modify and improve the software with relative ease.

Although intelligible to people and capable of being modified by them, source code does not operate computers directly. Before execution in a computer, source code must be converted into binary or object code for execution. This conversion is accomplished by computer systems running specialized computer programs designed for this purpose. The source code is the "source" of the binary machine instructions, which are derived from the source code through translation by a computer system running special programs.

There is a large number of programs intended to convert

---

9. Among the most common higher-level languages are the following: FORTRAN (short for "FORmula TRANslation"), used primarily for scientific and engineering applications requiring extensive algebraic computation; COBOL (for "COmmon Business Oriented Language"), used primarily for business applications including data base and information management; BASIC, a rudimentary language similar in some respects to FORTRAN, but having the advantage of ease of learning by non-technical personnel; and the "C" language developed by AT&T Bell Laboratories for use with its Unix operating system. Other common languages include ADA, developed and specified by the Department of Defense for military and defense applications, PASCAL, an elegant language useful for computer science and engineering research projects, and LISP (which has many dialects) for artificial intelligence and expert system applications.

10. For example, a simple program in BASIC designed to add the integers from one to ten and store the result in memory location "C", would read as follows:

```
10 C=0
20 DO 30 I=1, 10
30 C=C+I
40 STOP
50 END
```
source code into object code. The precise program used in conversion is determined by the language in which the source code is written and the computer and operating system under the control of which the program is to be converted or run. Conversion programs generally fall into two categories. "Compilers" are stand-alone programs that convert or "compile" source code into object code for storage and execution at a later time. "Interpreters" are programs that convert the source code to object code during operation of the program and, in effect, control the conversion process and operation of the program while the program is being converted and executed simultaneously. Most computer programs in use today are, or have been, converted into object code through the use of compilers.

The significance of the distinction between source code and object code is best illustrated by an example of software distribution. Suppose a software developer wishes to distribute its proprietary operating system to various computer manufacturers, who will in turn distribute the operating system, along with their computers, to end users. Since the operating system was not designed for any particular manufacturer's computer, in most cases it must be adapted to run on each manufacturer's special hardware. If the software developer does not wish to perform this adaptation, it must provide source code for the software to each computer manufacturer be-

11. Each model of central processing unit has its own "instruction set," that is, a set of correspondences between binary numbers and the computer operations that the binary numbers represent. For example, in one model of central processing unit, a particular number might represent the instruction for storing the contents of a central register in memory, whereas in another brand of computer, that same number might represent the instruction for adding the contents of a memory location to a central register. Each central processing unit also must run under the control of an operating system. This is a fundamental computer program that controls input and output operations, timing, operator monitoring and fundamental algebraic and logical operations for the computer. Since the object code produced by any conversion program must be consistent with the computer's instruction set and operating system, each conversion program must be designed for use with a particular instruction set and particular operating system.

12. The majority of existing computer programs are written in FORTRAN or COBOL. Although interpreters for these languages are available in some cases, historically these languages have used compilers. The majority of existing FORTRAN and COBOL programs in use today were developed with the aid of compilers.

13. Since an operating system performs fundamental input, output and control functions, it must be consistent with the instruction set of the microprocessor or other central processing unit, as well as with the architecture of the computer and its peripherals. For example, an operating system designed for use with Motorola's MC68000 microprocessor chip will not work with a computer designed around Intel's 8086 microprocessor chip. If the operating system is designed to be "portable," that is, adaptable to a number of different hardware environments, however, it may be a simple matter to modify the operating system to work with a new microprocessor or system architecture.
cause the manufacturers will not be able to adapt and modify the software using only object code.\(^{14}\) In many cases, manufacturers also may wish to have access to the source code so as to modify the operating system to make it conform to future modifications and improvements in their computers. As discussed below, the transfer of source code to these manufacturers usually involves complex licensing restrictions to protect the software developer's proprietary interests.\(^{15}\)

In contrast, when the manufacturer distributes its computers to end users along with the operating system, most end users do not need to modify the operating system. In the typical case, the end user simply uses the computer and operating system, and does not modify them. Accordingly, the computer manufacturer, under license from the software developer, can distribute only object code for the software to the end user without appreciably decreasing the marketability of the software or the computer.

The software developer normally insists that the computer manufacturer license the software only in object code form under these circumstances. Since object code is difficult, if not impossible, for people to understand and modify, an end user having only object code will be unable to alter the software or discern the ideas, techniques and algorithms that make it work.\(^{16}\) Thus distribution of the object code only provides practical protection of the software developer's proprietary interest at the end-user level, and may be an important element in providing legal protection as well.

The computer manufacturer, however, may wish to have the right to distribute the software in source code form to customers that want to modify or adapt the software by themselves, for example, to keep it compatible with modifications of the manufacturer's computer or to adapt it to other computer systems. The software developer and computer manufacturer must resolve these conflicting needs during negotiation of the license agreement. Of course, the resolution of this issue depends on the nature of the software and the computer and the needs of the marketplace, but the issue

\(^{14}\) Without source code the manufacturer wishing to adapt the software would not know which portions of the software perform which functions. Therefore he would not be able even to determine which portions require modification, let alone to do the modifications.

\(^{15}\) See infra pp. 50-51.

\(^{16}\) The end user would be able to understand and modify the software if the end user knew in which language the software was written and had access to a reverse compiling program for the end user's particular machine and operating system. In some cases, an end user can perform simple modification of input or output routines using only object code. In general, however, correction and modification of software without access to source code is prohibitively expensive and time consuming.
may be revisited later if an end user finds its needs cannot be met under an existing license arrangement.

Whether distributed in object code or source code form, computer software is easily copied. Unless the software developer or distributor takes technical precautions to prevent the software from being copied, the software may be copied, in usable form, with the aid of simple routines available on virtually any computer system. Although the cost of copying depends upon the size of the computer program and the recording medium, copying costs typically do not exceed a few dollars. The time required to copy a program normally is only a few minutes and seldom exceeds an hour.

Computer software thus is a valuable product that can be duplicated and stolen at minimal cost. Indeed, recent commentators have estimated that as much as thirty or forty percent of the value of software used in the United States today may be "bootleg" software, copied and used without the authority of the owner. To protect their investments in research and development, most software developers turn to technical means of protection and ultimately to the law.

B. Patent Protection

Under American law, a patent provides the strongest protection of intellectual property. It grants an absolute monopoly to make, use and sell the patented product or process for a period of seventeen years. Unlike copyrights and trade secrets, it protects against independent creation of the subject matter. In theory, the sword and shield of patent protection would be useful tools to protect the investment and effort in development of software.

In practice, however, patent protection is inappropriate for

18. Copying time depends on the length of the program and the medium on which it is recorded. Copying of floppy diskettes used for most microcomputer programs takes at most a few minutes. With proper equipment, copying of major programs for mainframe computers recorded on magnetic tape can be accomplished in less than half an hour.
21. A copyright protects only against copying, not against independent creation of the same material, or even copying from a common source in the public domain. See, e.g., Selle v. Gibb, 741 F.2d 896 (7th Cir. July 23, 1984); M. Nimmer, Nimmer on Copyright § 201[A] (1984). Trade secret law protects only against misappropriation of the trade secret from its owner, not against independent creation or reverse engineering. See Restatement of Torts, § 757 (1939); Unif. Trade Secrets Act, §§ 2, 3; R. Milgrim, Trade Secrets §§ 2.05[2]; 504[1] (1984).
computer software in most cases. Under a longstanding line of author- 
ity affirmed repeatedly by the United States Supreme Court, 
patents are not available for algorithms and mathematical for- 
mae. For example, in Gottschalk v. Benson, the Supreme Court 
held that a commonly used algorithm for converting binary-coded 
decimal numbers into binary numbers was not patentable. Algo-
rithms and mathematical formulae, however, are the essence of 
most computer programs. The unique portions of any computer 
program are primarily algorithms and techniques like the one dis-
cussed in Benson, some of which are generally known and some of 
which are not.

This is not to say that no software is patentable. The Supreme 
Court has never closed the door entirely on software patents. In 
Diamond v. Diehr, the Court upheld a patent for a process for 
curing rubber in which computer calculations of the precise time at 
which to open the mold were the essential inventive element. More 
recently, Merrill, Lynch, Pierce, Fenner & Smith, Inc. negotiated a 
$1 million settlement of its claims that other brokerage houses had 
infringed its patented cash management system. The Supreme 
Court's decisions, however, have left considerable uncertainty 
whether software is patentable by itself, that is, apart from a system 
including hardware or a process involving significant “post-solu-
tion” activity.

In any event, legal uncertainty whether software is proper sub-
ject matter for patent protection is not the only problem with patent 
protection for computer software. The patent system imposes high 
standards of originality and inventiveness upon items subject to its 
protection. In order to be eligible for patent protection, an inven-
tion must be both “new” — that is, not revealed in certain patents 
and printed publications — and not “obvious” to an ordinary per-
son skilled in the art to which the subject matter pertains.

These standards are inappropriate for computer software for 
three reasons. First, due to the widespread use of computer 
software and the rapid pace of its development, it is virtually impos-
sible to determine whether any particular twist or technique in com-
puter software is “new.” Computer software is used and/or

63, 71-72 (1972).
developed by or for almost every business and industry in the United States and throughout the world.\textsuperscript{28} Even if a centralized library of all this material existed, it would be a prodigious task to search all exemplars of a particular type of software to determine whether a particular algorithm used in it was “new.”

Secondly, most of the original content of computer software consists of small step-by-step advances, which are generally too small to meet the statutory standard of inventiveness, or the use of known algorithms in new ways or to solve new problems — an application not eligible for patent protection under black letter patent law.\textsuperscript{29} Nevertheless, the development of computer software demands massive investment in human effort and money\textsuperscript{30} which would not be made without some form of reliable legal protection from piracy.

Finally, the task of determining whether a particular new twist in computer software is “nonobvious” is metaphysical at best. The “nonobviousness” standard, originally developed for patents on mechanical devices, has caused more than one judge to throw up his hands in assessing chemical and electronic claims.\textsuperscript{31} The task is all the more difficult in a field governed by purely abstract mathematical invention, in which many small ingenious advances are made by many people every day, but those advances are likely to be classified as mathematical formulae or procedures that are not appropriate subject matter for patent protection. Thus in the field of computer software it is easy to confuse the distinct legal issues of what is patentable subject matter and what advances within a generally patentable field are nonobvious enough to merit a patent.

\textsuperscript{28} For a good discussion of the widespread use of computer software in business and industry, see \textit{Bus. Week}, Feb. 27, 1984, at 74.

\textsuperscript{29} \textit{See supra} note 22.

\textsuperscript{30} A computer program is like an elaborate instruction manual intended to communicate with machines, rather than with people. Its value lies not in the uniqueness or “nonobviousness” of its design, but in the fact that it was designed at all. Although not requiring great ingenuity or genius, the writing and debugging of computer programs does require intensive human effort, often measured in person-years. Without protection from piracy, there would be no incentive for business to incur the expense to underwrite this effort. Once a program is properly written, however, its value lies in its ability to perform mathematical calculations or to manipulate information with a speed and accuracy that could not be achieved by unaided human effort.

\textsuperscript{31} \textit{See, e.g., Parke-Davis & Co. v. H. K. Mulford Co.}, 189 F. 95, 115 (C.C.S.D.N.Y. 1911) (L. Hand, J.), aff'd in part and rev'd in part, 196 F. 496 (2d Cir. 1912) ("How long we shall continue to blunder along without the aid of unpartisan and authoritative scientific assistance in the administration of justice no one knows. . ."); Picard \textit{v. United Aircraft Corp.} 128 F.2d 632, 639 (2d Cir.) (Frank, J., concurring), \textit{cert. denied}, 317 U.S. 651 (1942) (Patent affairs need "the judgment of men who are experts in science.").
A further reason why patent protection is not appropriate for computer software is that it is too expensive and slow. One reason for the rapid development of the software industry is that it is labor-intensive but requires little capital equipment. Many advances in software have been made by professors, students or children using inexpensive personal computers or time-sharing systems with little capital investment. For these individuals, the cost of patent filing and maintenance fees is significant, and the cost of legal fees for prosecution of a patent through several office actions in most cases would be prohibitive.

As for speed, delays in achieving patent protection often exceed the useful lifetime of software products. For example, it is not uncommon for uncontested patent applications to take two years to prosecute, and any patent interference action may add significantly to that delay. Much software, however, becomes obsolete in seven years, and, in the mushrooming and highly competitive

---

32. Programmers can buy time on sophisticated university or commercial time-sharing computers at reasonable rates. Today, personal computers are available for home use at modest cost. For example, a personal computer having hard-disk storage and higher-level language capability costs less than $5,000.

33. See, e.g., Bus. Wk., Feb. 27, 1984, at 77 (Ashton-Tate, a well-known software developer and distributor, founded on $7,500); FORTUNE, Apr. 30, 1984, at 200 (teacher and part-time music composer writes successful software); ELECTRONICS, June 16, 1983, at 54 (UPI network software developed by graduate student); INFOWORLD, Feb. 6, 1984, at 82 (discussion of professionals from noncomputer fields as amateur programmers).

34. For individuals, small businesses and nonprofit organizations, the patent filing fee is a minimum of $150, but it can be considerably higher, depending upon the nature of the patent claims. The patent issue fee is $250. 37 C.F.R. §§ 1.16, 1.18 (1984). Maintenance fees for individuals, small businesses and nonprofit organizations are $200, $400 and $600, due at the end of the fourth, eighth and twelfth years after the patent issues, respectively. See 37 C.F.R. § 1.20 (1984). If the software developer is the user of a $5,000 personal computer, filing, issue and maintenance fees alone will exceed one-third of the developer's capital investment in the software business.

35. Patent prosecution fees of course depend upon the nature of the patentable invention, the state of the prior art and the response of the patent examiner. However, fees of competent attorneys for a patent search, analysis of the patent search, initial drafting of the patent application (including draftpersons' fees for drawings), and response to the examiner's first office action are unlikely to be much less than $5,000. Thus, the hypothetical entrepreneur with a $5,000 personal computer would have to commit to doubling his capital investment in the business simply to attempt patenting a computer program, without any guarantee that a patent would issue, or, if issued, would be enforceable.

36. In August 1984, Commissioner of Patents and Trademarks Gerald J. Mossinghoff stated that it took slightly more than two years, on the average, to get a patent. The Patent and Trademark Office's goal is to reduce this by two months in each of the next three years to achieve an eighteen-month average delay by 1987. 28 PAT., TRADEMARK & COPYRIGHT J. (BNA) No. 692, at 436 (Aug. 16, 1984).

37. A 1982 study showed that the average duration of patent interference actions was two years, but some took more than nine years. 64 J. PAT. OFF. SOC'y 699, 700 (1983).

38. The author's clients generally recognize a seven-year "rule of thumb" for software
personal computer software industry, products may lose their primary value in an even shorter time.\(^ {39} \)

The final reason why patent protection is not appropriate for software is that patents are difficult to enforce. The bulk of the vast amount of computer software used in the world is hidden in magnetic media and computer storage systems. There is no practical way for a software vendor to determine whether any particular user, or even a large class of users, possesses or is using "bootleg" copies of that vendor's software. Under these circumstances, most vendors are reluctant to explain how their software operates in a patent application which, although kept secret during the application process,\(^ {40} \) is required by law to be disclosed to the public once the patent issues.\(^ {41} \) For many vendors, the "enabling disclosure"\(^ {42} \) that becomes part of the public records of the Patent and Trademark Office is tantamount to a license for piracy.

C. Copyright

Of the three traditional pillars of intellectual property protection, copyright is the most appropriate for computer software. It protects against copyng — that is, direct piracy — without prohibit-

obsolescence. The useful lifetime of a particular piece of software, however, depends upon the nature of the software and the machine on which it is designed to operate. Operating systems designed for particular machines will have the same useful lifetime as the machines themselves. Moreover, some fundamental programs have relatively long lifetimes. For example, variants of the original FORTRAN compilers developed in the 1950s are still in use.

39. Short lifetimes are the general rule in the personal computer industry due to rapid advances in both hardware and software. For example, the well-known spreadsheet program "VisiCalc," the first spreadsheet program and one of the most successful programs for personal computers, was first introduced in 1979 and by 1984 was losing significant market share to more versatile programs containing other applications software in addition to spreadsheets, such as Lotus 1-2-3. \textit{See Bus. Wk.,} Feb. 27, 1984, at 88. Less successful programs, as well as those in fields crowded with competition, often have even shorter marketing lives. \textit{See Shakeout supra} note 1 at 269 (commercial viability of typical personal computer software is about one year).

40. The patent statutes require that the disclosures in patent applications be kept secret until the patent issues. \textit{See 35 U.S.C.} § 122 (1982). This allows the patent applicant to maintain trade secret protection for the subject matter of the invention if a patent does not issue. If a patent issues, the disclosure becomes accessible to the public in the files of the Patent and Trademark Office, and anyone is permitted to practice the invention after the end of the patent term.


42. The disclosure included in the public records of the Patent and Trademark Office is called an "enabling disclosure" because it is required by statute to contain sufficient information to permit a person having ordinary knowledge and skill in the art to which the patent pertains to practice the invention using information contained in the disclosure. \textit{See 35 U.S.C.} § 112.
ing independent creation and thereby "chilling" development of the industry. Although certain technical requirements of copyright protection may be cumbersome, copyright protection is in one sense automatic, and full protection is cheap and easy to obtain. Indeed, in one sense copyright protection is overkill, for its duration — fifty years at a minimum — is significantly longer than the foreseeable useful life of most computer software in operation today.

Nevertheless, copyright protection has significant gaps that makes its coverage of software incomplete. First, based on the line of authority beginning with the seminal decision of Baker v. Selden, and confirmed in the language of Section 102(b) of the Copyright Act, copyright protects only expression, and not the underlying ideas. Since software is designed to operate machines, however, its expression is not as important as its underlying function, that is, the ideas expressed in the code.

The underlying function is, of course, protected indirectly by copyright to the extent that copyright protects not only against verbatim copying, but also against substantial copying of organization, theme, incident, sequence and detail. But pure ideas, algorithms and mathematical procedures are not traditionally the subject of copyright, and it is doubtful that courts would or should protect them under the aegis of copyright, especially if they can be expressed in only a limited number of ways or are expressed in a

43. For copyrights owned by individual authors, the term of protection is the author's lifetime plus fifty years. 17 U.S.C. § 302(a) (1982). For anonymous and pseudonymous works and works made for hire, the term of protection is seventy-five years from first publication or one hundred years from creation of the work, whichever expires first. 17 U.S.C. § 302(c) (1982).
44. 101 U.S. 99 (1879); see 1 M. Nimmer, Nimmer on Copyright §§ 1.10[B][2], 2.03[D] (1984).
46. Copyright protection varies along a spectrum ranging from full protection of exact expression at one end, to no protection for fundamental ideas, methods and processes at the other. Between these two extremes, the degree of copying of organization, theme, incident, sequence and detail must be examined according to the "levels of abstraction" test enunciated by Judge Learned Hand in two famous copyright decisions: Sheldon v. Metro-Goldwyn Pictures Corp., 81 F.2d 49, 54-56 (2d Cir. 1936); Nichols v. Universal Pictures Co., 45 F.2d 119, 121 (2d Cir. 1930). See generally 1 M. Nimmer, Nimmer on Copyright § 1.10[B][2] (1984).
47. See supra sources cited in note 44.
48. If the number of ways in which a certain function can be expressed is too limited, however, copyright protection may be weak or nonexistent. The copyright laws do not permit an author to preempt ideas or functionality that are capable of expression in only a limited number of ways by retaining exclusive rights in one or more of the limited ways of expression. See, e.g., Sid & Marty Krofft Television v. McDonald's Corp., 562 F.2d 1157, 1168 (9th Cir. 1977); Continental Casualty Co. v. Beardsley 253 F.2d 702, 704-06 (2nd Cir. 1958).
form different from that of the original, for example, by translation into a different computer language using a different computer operating system with different architecture and different instruction sequences. Since functional ideas and their elaboration constitute one of the chief and most difficult objectives of software development, copyright alone can never be more than a half measure of protection for the software developer.

The second reason why relying solely on copyright protection is terrifying for software vendors is the "first-sale" doctrine. Under this doctrine, enshrined in Section 109 of the Copyright Act, anyone who owns a copy of a copyrighted work has the right to sell, transfer, lease and otherwise dispose of that copy at his discretion, without accounting to the copyright owner. Literally interpreted, this provision would permit a business, without the permission of software developers, to buy copies of their software and rent them out to users on a short-term basis, who would then have the opportunity, if not the right, to make bootleg copies for their own use.

To avoid this situation, software developers usually distribute their software only pursuant to written licenses that contain certain protective provisions. Some software licenses state explicitly that the end user does not obtain ownership of the copy but only a license to use it. Other license agreements obligate the end user not to transfer or sublicense his copy of the software, attempting to contravene by contract the permission granted by the statute. Such contractual provisions are currently the subject of a legal action in the Northern District of California, whose outcome may depend on the enforceability of "box top" software licenses unsigned by the end user.

Retention of ownership of the copy is problematic in light of the facts that the end user retains possession of the copy and that many states have statutory and common-law policies against restraints on alienation of personal property. Contractual prohibitions against transfer and sublicensing of the copy are sounder

49. This doctrine holds that the copyright owner legally loses control over copies of the copyrighted work after the first sale of a copy to a purchaser. As long as the purchaser does not make additional copies of that work the purchaser is free to deal with the copy purchased as he or she wishes. See 1 M. Nimmer, Nimmer On Copyright § 8.12[B][1] (1984).


51. Micropro Int'l Corp. v. United Computer Corp. (N.D. Cal. C-83-3019-WWS 1984); see also Russo, supra note 4.

because the legislative history of the Copyright Act indicates that contractual restrictions on the statutory permission are valid, although they may not be enforced by copyright infringement remedies. Nevertheless, both protections rely upon an enforceable agreement between the copyright owner and the end user, and this agreement may be unenforceable where, as often occurs in personal computer industry, it is an unsigned agreement that the end user reads only after purchasing and preparing to use the software product.

It is not the purpose of this article to discuss the arguments pro and con enforceability of “box top” software licenses. Suffice it to say that the issue has generated enough uncertainty and concern to result in the passage of corrective legislation in Louisiana and the preparation of a similar bill in California. The salient fact is that the first-sale doctrine, in the view of most software vendors and their attorneys, requires significant modification by contract in order to permit business to be done in the ordinary course where software is concerned.

Finally, a number of the technical requirements of the American Copyright Act are inappropriate and cumbersome as applied to computer software. Copyright notices are required on “published” works, not on “unpublished” works. It is difficult, however, to determine whether a particular copy of software has been “published.” If software is distributed widely on diskette without any accompanying license, “publication” might be an appropriate concept. However, this is seldom done. In most cases, software is distributed under written license agreement to a limited group of users—that is, to users who have the necessary computer equipment and operating system on which the software runs. Is this “publication”? This question has no answer. Any attempt to draw the line based on the number of copies distributed would have the absurd result, under the current Copyright Act, that a vendor who began distribution to a limited group without using a copyright notice would have to try to put notices on all previously distributed copies after distribution reached a certain level.

---

54. See generally, Russo, supra note 4, for a discussion of the enforceability of “box top” software licenses.
57. Once a work is “published” under the Copyright Act, it should contain copyright
A second set of technical requirements that are inappropriate for computer software consists of the requirements for registration and deposit. These requirements are inappropriate for two reasons. First, most software is constantly modified in the process of marketing and distribution, especially in the early stages of development. Modifications are required to fix "bugs," to maintain compatibility with changing hardware and operating systems, and to adapt to new hardware and operating system environments. Technically, the Copyright Act requires that each new version with significant original material be registered as a separate work, but this is administratively impractical.

Secondly, the Copyright Office's requirement for deposit of source code in the public records has raised the specter of piracy. The Copyright Office has promulgated interim rules permitting deposit of identifying materials only and is in the process of developing a final rule. Even this reduced requirement, however, appears to be futile in light of the Copyright Office's own admission that: (1) it cannot store or retain all the material it receives; (2) it cannot verify whether what is deposited is original material; and (3) to the extent the identifying material does not expose significant portions of software, the deposit does not serve its obvious purpose of acting notices. If notices are omitted from more than "a relatively small number of copies" of the work distributed under the authority of the copyright owner, the copyright owner must register the work within five years after publication and make reasonable effort to add the notice to copies distributed without notice. 17 U.S.C. § 401(b)(2) (1978).


59. Infringement actions cannot be maintained until after the copyright in the allegedly infringed work is registered. 17 U.S.C. § 411. In addition, statutory damages and attorneys' fees are not available in an action for acts of infringement occurring before registration, except during a three-month grace period immediately after first publication. See 17 U.S.C. § 412.

60. See generally Wilbur, Copyright Registration for Secret Computer Programs: Robber of the Phoenix's Nest, 24 JURIMETRICS J. 357 (1984). The Copyright Office will accept object code for deposit, but only subject to the "rule of doubt." See 48 Fed. Reg. 22951, 22952 (May 23, 1983). A letter invoking the "rule of doubt" attached to the registration certificate issued by the Copyright Office, as a practical matter, undermines the value of the registration certificate in litigation. For a "clean" registration unsullied by the "rule of doubt," the Copyright Office requires deposit of source code for at least portions of a computer program. See id.; 37 C.F.R. § 202(c)(2)(vii) (1983).

as an evidentiary basis for copyright infringement actions. The requirement, however, does impose a burden upon software developers in the form of administrative expense and the attorneys' fees they must pay to determine which portions of the code can or must be submitted to maintain copyright protection without risking undue exposure of underlying trade secrets.

D. Trade Secrets

Although the technical requirements of copyright protection are burdensome, they can be satisfied through the implementation of appropriate procedures, albeit at added administrative and legal expense. The major deficiency of copyright protection is not its technical requirements, but its failure to protect ideas. This fact is responsible for the increasing attention that lawyers representing software developers have paid to trade secret protection over the last several years, despite the increasing weight that the courts have given to the principles of software protection under copyright. For it is the ideas that are technically ineligible for copyright protection — the algorithms, mathematical formulae, calculational procedures, methods and techniques — that are the essence of computer programming. Moreover, these ideas are not well protected by copyright because they often can be expressed in a number of different ways and their expression can be transferred from one form to another easily — often by machine.

In order to protect those elements of computer software that are not eligible for copyright protection, vendors must invoke the law of trade secrets. A “trade secret” is any


63. Rearranging the sequence of the computer program itself to preserve trade secrets might be viewed as an improper practice, weakening the copyright registration or leading to unfavorable inferences in litigation. The Copyright Office's informal interim rules, however, provide three alternative means for making a deposit. See supra note 61. One of these means permits the copyright owner to obliterate up to 50% of certain required deposit material. Thus the choice of means, as well as the choice of material to delete if this particular means is chosen, gives the copyright owner some discretion concerning what to disclose.

64. Until two years ago, many large computer vendors protected object code for their computer programs almost entirely by copyright. In early 1983, IBM announced restrictions on distribution of source code for its operating systems and other computer programs and began to include covenants against reverse engineering of object code in its software licenses. See ELECTRONIC NEWS, Feb. 1, 1983, at 1. Following IBM's lead, most of the computer industry is now paying greater attention to protection of trade secrets in computer software in both source code and object code forms.

65. See, e.g. Apple Computer, Inc. v. Formula Int'l, Inc., 725 F.2d 521 (9th Cir. 1984); Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983); GCA Corp. v. Chance, C-82-1063-MHP, 217 U.S.P.Q. 718 (N.D. Cal. 1982).
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. It may be a formula for a chemical compound, a process of manufacturing, treating or preserving materials, a pattern for a machine or other device, or a list of customers.  

According to this definition, almost any aspect of the computer program, including the underlying ideas and the entire program itself, is eligible for trade secret protection. Only three requirements must be met: (1) the protected element must have competitive value; (2) it must not be generally known to the public or (although this is less certain) to the computer industry; and (3) the owner must take reasonable steps to keep it secret.  

At first blush, trade secret doctrine would appear to have a number of significant advantages for computer software protection. First, the scope of protection is very broad. Almost anything can qualify as a trade secret, according to the Restatement of Torts, except transitory information such as a market price or a particular quote or bid. Thus the software developer might protect as a trade secret an algorithm, a computational procedure, a subroutine, or an entire computer program. Secondly, like copyright protection, trade secret protection does not prohibit independent creation and therefore erects no barriers to further development of the software industry. Thirdly, trade secret protection has potentially infinite duration; it lasts as long as its subject matter is kept secret and is not generally known. Finally, no special legal procedure or application to a government agency is necessary to secure trade secret protection. Provided the proper measures are taken to insure secrecy, protection subsists from the moment the trade secret is discovered or implemented. There are no filing fees, administrative routines or legal expenses required to obtain protection of a particular trade secret, although administrative and legal expenses may be incurred in implementing and maintaining a program for the protection of trade secrets in general, or in enforcing rights in any

---

68. See Restatement of Torts § 757 comment b (1939).  
69. See supra note 21.  
71. In order to protect its trade secrets, a business must take reasonable effort under the
particular trade secret. Given all these advantages, trade secret protection would appear to fill the considerable gap left by patent and copyright protection at insubstantial cost to the software developer and vendor.

III. DIFFICULTIES WITH TRADE SECRET PROTECTION

A. Filling the Gap

As discussed in the previous section, the three traditional pillars of intellectual property law do not support a protective roof for computer software. Copyright protection — in some ways the most appropriate form of protection for computer programs — protects only the expression of mathematical ideas in particular code, not the underlying ideas and procedures. This protection is generally insufficient, for it is nearly always possible to express the underlying procedures and ideas in many different ways. Indeed, since computer programs are written in highly stylized form and may be handled by machines, it may be possible to write computer programs that would translate, reorganize and modify other computer programs automatically in such a way that copyright infringement would be difficult to detect, if not to prosecute as a matter of law. Thus copyright does not protect the functional aspects of computer programs that are their most important elements.

On the other hand, patent law, the traditional domain of protection of function, is inaccessible to most software vendors due to the high standards of protection and the expense and delay involved in obtaining patents. Those who invest in research and development in computer software would like all of their computer programs protected, not just those, or elements of those, that satisfy the high standards of novelty and nonobviousness necessary to obtain a patent. Moreover, the underlying mathematical ideas and algorithms, if not entire programs, are unpatentable. If one adds to these uncertainties the poor batting average of patents when their validity is challenged in court and the necessity for revealing all to circumstances to maintain secrecy. Most businesses have trade secret programs reviewed by counsel and higher management. These programs prescribe and implement such precautions as security guards, visitors' permits, plant access restrictions, locked areas and file cabinets, document classification and destruction and restraints on technical publication in order to preserve trade secrets.

73. See supra pp. 36-37.
74. See supra pp. 34-35.
75. Depending upon the forum, the majority of patents are invalidated on one ground or another when challenged in court. See FORBES, Sept. 10, 1983, at 163.
the public at large when a patent issues, it is not surprising that few software vendors seek patent protection for their programs.

Since copyright protection does not apply to ideas or algorithms and patents are of little help in practice, trade secret protection must fill the gap. By now, most software vendors have recognized the need for trade secrecy as an adjunct to copyright protection. Yet, despite the advantages enumerated above, trade secret law is an imperfect tool that often impedes, rather than aids, the flow of commerce in software.

B. The Paradox of "Secret Software"

The primary difficulty with trade secret law as applied to software is that it is a contradiction in terms. Trade secrets traditionally have been specific formulae, processes, or items of information that are kept carefully locked within a plant or business and never see the light of day. On the other hand, a software marketer's primary goal is to market and distribute his software as far and as wide as possible. This paradox is reflected in several unresolved legal issues relating to trade secret protection for software.

The first troublesome issue is the question of "publication." Traditional trade secret principles hold that, once a secret is "published," it thereby becomes known to the general public and legal protection is lost. Yet most software developers distribute their software as widely as possible for profit and, in so doing, advertise that they will supply their software to anyone who wants it. Moreover, in order to be sure that copyright protection is not lost, most software developers affix copyright notices to copies of their software that they distribute. Since copyright notices are required only on works that are "published" within the meaning of the Copyright Act, one can argue that affixation of a copyright notice is an implied admission that the software has been published.

Use of a copyright notice, however, does not necessarily imply publication of software, even for purposes of the Copyright Act. The Copyright Act does not prohibit the use of the copyright notice

76. See supra p. 37 & note 42.
79. This argument, however, has not fared well in the courts. See, e.g., Technicon Medical Info. Sys. v. Green Bay Packaging, Inc., 687 F.2d 1032, 1036-39 (7th Cir. 1982), cert. denied, 74 L. Ed. 2d 955 (1983); M. Bryce & Assoc., Inc. v. Gladstone, 319 N.W. 2d 907, 913 (Wis. 1982).
on unpublished works,\textsuperscript{80} and software vendors are well advised to include copyright notices even on unpublished works to preserve their copyright in the event that a court later decides there has been publication. Moreover, the scope of "publication" for copyright purposes is not the same as the scope of "becoming publicly known" for purposes of loss of trade secret protection. At least two courts have upheld trade secret protection for distributed software on this ground.\textsuperscript{81}

Nevertheless, there remains the question whether widely-distributed software is or should be entitled to trade secret protection. Under current law, the answer is uncertain. In one case, a court upheld trade secret protection for design information in confidential minicomputer maintenance manuals that the defendant claimed were distributed to almost six thousand persons.\textsuperscript{82} This number, however, is small compared to the customer lists of many software vendors. In any event, the precise number should have little bearing upon whether the software has become generally known in the industry so that trade secret protection has been lost.

Suppose that there are one thousand potential users of a given software package, and that all but one have received and paid for the package under license from the vendor. Assuming that the one thousand users define the relevant industry, the software would be generally distributed within that industry if the word "generally" has any meaning. Nevertheless, it would be both unfair and economically unsound\textsuperscript{83} for the last potential user to be permitted to use the software without paying the vendor the license fee that all other users in the industry have paid. Thus, for software that is licensed and not sold, the principle that trade secret protection is lost when something becomes distributed should not apply, unless the words "generally distributed" are construed to mean "distributed to all" under these circumstances.

More fundamentally, one can argue that having been generally distributed should not be construed as being generally known where licensed software is concerned. The rule against trade secret protection for general knowledge is designed to prevent claims of exclu-

\textsuperscript{80} 1 M. Nimmer, Nimmer on Copyright, §§ 4.01, 4.04 (1984).
\textsuperscript{81} Supra note 79.
\textsuperscript{83} Not only would the last potential user be getting a "free ride," but such a rule would encourage all potential users to delay licensing the software, or perhaps even to infringe, in the hope that they would be held to be among the lucky few who did not have to pay for a license.
sive property rights in what lies in the public domain. When software is licensed, however, it is not the ideas and underlying technology in the software that are licensed but the right to use the software. When a vendor licenses the use of a computer program containing a clever algorithm for computing accounts receivable, for example, he does not intend ordinarily to permit the licensee to use that algorithm to construct another program, but rather to permit the licensee only to use that algorithm in the vendor’s program to calculate accounts receivable. Accordingly, the dilemma regarding the degree to which widespread licensing may occur before trade secret protection is lost can be resolved by recognizing that it is not the trade secrets themselves that are licensed, but their use.\footnote{This recognition is fundamental to an appreciation of the role of software in intellectual property. Software's value differs from that of most copyrighted works because it inheres not in what the software communicates to people, but in the functionality of the software in controlling machine processing. In this sense, software is less like a copyrighted instruction manual and more like the subject matter of a patent.}{44}

Whether the law will make this fine distinction, however, remains to be seen. In the interim, owners of software must tolerate considerable uncertainty and, if legal protection is desired, must negotiate appropriate licensing agreements at every stage of the distribution chain, as discussed below.\footnote{See infra pp. 50-52.}{55}

C. Reverse Engineering

As mentioned above,\footnote{See supra pp. 31-33.}{66} one of the methods that software distributors often use to protect their proprietary interests in software is “object-code-only” licensing. Under this scheme of licensing, the vendor distributes recording media containing only object code and makes all modifications and corrections to the software itself, using the source code that only it retains. This form of licensing has two effects. First, since object code is difficult, if not impossible, to understand and modify, it provides practical, technical protection against misuse of the software and the ideas hidden in the object code. Secondly, because object code is difficult, if not impossible, to understand by itself, this form of licensing might provide some legal protection. The software vendor could argue that, by releasing object code only, it has taken all reasonable steps necessary to protect the ideas and trade secrets embedded in the software.

Under current law, object-code-only licensing should not provide legal protection against the lawful owner of a copy of the software in the absence of contractual prohibitions against reverse
Black letter trade secret law allows the owner of personal property to examine, dissect and disassemble it in any manner and to use freely any information obtained in so doing.\footnote{See, e.g., Colony Corp. v. Crown Glass Corp., 430 N.E.2d 225, 227 (Ill. App. 1981); Analogic Corp. v. Data Translation, Inc. 358 N.E.2d 804, 807 (Mass. 1976); A. F. Holden Co. v. O'Brien, 73 U.S.P.Q. 481 (E.D. Pa. 1947); see generally, Restatement of Torts § 757 comment a (1939); R. Milgrim, Trade Secrets § 2.05(2) (1984); Grogan, Decompilation and Disassembly: Undoing Software Protection, 1 The Computer Law. 1 (1984).} The source of this doctrine is the notion that the owner of property should be able to treat it in any way he or she chooses.

Application of this doctrine does not end the matter, however. If an end user is a licensee of the software, and not an owner of a copy of it, the license agreement should control. At least, the end user should have no more right to reverse engineer and use the ideas in the software than are explicitly granted in the license. To this extent, the law of trade secrets is analogous to the "first-sale" doctrine in copyright law,\footnote{See supra pp. 38-39.} under which the issue of ownership of a copy determines the rights of the user freely to dispose of or lease that copy. The Copyright Act's legislative history permits the effect of ownership of a copy of the software to be varied by contract.\footnote{See supra p. 40.}

Over the past year or two, many large software vendors have attempted to take a similar tack with respect to trade secrets by inserting in their written license agreements prohibitions against reverse engineering, reverse compiling, and disassembling the software.\footnote{Although a number of smaller vendors have done this on the advice of counsel for several years, IBM was the first of the major computer vendors to include these clauses in its agreements. Supra note 64.} There appears to be no good reason why these prohibitions should not be enforceable under contract law, just as the Copyright Act permits enforcement of contractual prohibitions or limitations on a copy owner's transfer and licensing rights. The enforceability of these prohibitions, however, would be conditioned upon existence of a valid contract — a prerequisite not necessarily present when "box top" licenses are used.\footnote{The enforceability of "box top" software licenses under state law generally depends on the circumstances under which the software is furnished. If the licensee has an opportunity to read and review the license before the licensing transaction is completed, the "box top" license may be enforceable. Under some circumstances common in commercial practice, however, the license agreement remains hidden until the licensing transaction is completed and the licensee unpacks the software and/or hardware to begin using it. Under these circumstances, there are strong arguments why the "box top" license should not be enforced. See generally, Russo, supra note 4.}

Enforceability of contractual restrictions on reverse engineer-
ing is not only consistent with the outline of current copyright law, but appears to be good policy. Growth and development in the software industry are rampant; barriers to entry appear to be low.\footnote{As discussed above, see p. 36, supra, capital investment required for entry is minimal. This may be changing, however, at least in the consumer software marketplace. Recent reports indicate that substantial advertising expenditure may be necessary successfully to market products for personal computers. \textit{Fortune}, Sept. 19, 1981, at 59 (for the larger vendors of software for personal computers, marketing is now single largest expense, exceeding research and development); \textit{Bus. Wk.}, Feb. 27, 1984, at 74, 77 (requirement for large advertising budgets has become barrier to entry into microcomputer software business).} There is no good reason to permit a licensee of software to make use of the licensor’s trade secrets buried in the object code by reverse engineering that code if the license is explicitly restricted to use only. Moreover, if programs that reverse compile object code do exist, they are likely in the hands of only the largest and most sophisticated software houses. Invalidating restrictions on reverse engineering would benefit only these large businesses and increase concentration in the industry. In any event, the protection of contractual prohibitions on reverse engineering require negotiation as part of the license agreement; otherwise the software licensor is subject to the uncertain application of the common law.

D. "Reasonable Steps"

A final area of uncertainty in the application of trade secret law to software is the question of what reasonable measures must be taken to preserve trade secrets in the process of licensing and distribution. Traditional trade secret law provides that whatever steps are taken must be reasonable under the circumstances.\footnote{See generally \textit{Restatement of Torts} § 757 comment a (1939); \textit{Unif. Trade Secrets Act} § 1(d)(2) (1979); R. Milgrim, \textit{Trade Secrets} § 2.04 (1984).} Unfortunately, this general standard is sufficiently vague to create considerable uncertainty, and the uncertainty often causes software lawyers to take conservative positions in license negotiations.

License agreements typically contain one or more of three standards for nondisclosure of trade secrets. First, there may be an absolute requirement that the software be kept confidential and not disclosed. Secondly, the licensee may be obligated to use "best efforts" or "reasonable efforts" to preserve secrecy. Finally, the licensee may be required to use the same care in handling the licensed software as it uses in handling its own software. Often, the second or third standard are imposed together, in the conjunctive.

Whatever verbal formula is used, its legal effect may depend upon the circumstances. For example, if a vendor imposes the
"same care" standard upon a major, sophisticated software developer, that may be sufficient to preserve trade secrecy. However, it may not be sufficient if the software vendor knows that the licensee has a policy of not maintaining adequate protection of software that it licenses from others, or that the licensee is an unsophisticated new entrant to the software business that does not have trade secret protection measures in place.\footnote{Since the test is what is reasonable under the circumstances, a court could find that the vendor's knowledge of the licensee's inadequate protective procedures or general naivete in business matters made reliance on the "same care" standard unreasonable.}

Although the "reasonable care" standard should be adequate in most cases, it may be insufficient if the licensor knows that the licensee has a history or policy of inadequate care. Even the absolute obligation to maintain secrecy may not be enough if the licensee flouts that obligation and the licensor takes no remedial action. Since all of these circumstances are to be reviewed by a court after the fact, there can be little certainty regarding trade secret protection under any standard. As a consequence, many software lawyers negotiate hard for the highest standard, and some even seek indemnification for breach of the standard of care.

E. Licensing Negotiations

The result of all this uncertainty can be a complex chain of lengthy and interlocking license negotiations. For example, consider the situation discussed above in which an operating system developer licenses the operating system in source code form to manufacturers of computers that distribute the operating system in object code form with their computers. If the operating system vendor is distributing the source code for its major product, it normally will treat that code like the "family jewels" and attempt to impose stringent restrictions on use and disclosure upon the manufacturers. These restrictions may include an individual, signed confidentiality agreement with each employee of each manufacturer who has access to the source code.

For the reasons discussed above, the vendor also will wish to control to some extent licensing at the second level from the manufacturer to the end users. For example, to preserve its trade secrets, it will wish to control the standard of care imposed upon the end user to prevent unauthorized disclosure of the software. In addition, it may wish to force the manufacturers to include in end-user licenses prohibitions against reverse engineering of the object code.\footnote{See supra pp. 47-49.}
and, to avoid application of the first-sale doctrine, restrictions on transfer and licensing or statements that the end user is not the owner of a copy. 96

The manufacturer, of course, will resist the imposition of these conditions for several reasons. First, the manufacturer already may have form license agreements in place that it does not wish to change because changes would entail additional printing and administrative expenses and might generate anxiety among customers. Secondly, the manufacturer may view the software as part of a hardware system sold by the manufacturer and may resist using any software license agreement. Thirdly, the manufacturer will wish to impose minimum restrictions on its customers in order to maximize the marketability of its hardware and lower its cost of administration. The end result is often difficult negotiation over terms of the vendor/manufacturer license agreements that are often ignored in practice by all but the lawyers. 97

The foregoing example involved only three levels of distribution: the vendor, the manufacturer and the end-user. Even for three levels, separate license agreements, including provisions governing end-user licenses, would have to be negotiated between the software vendor and each manufacturer of the computer systems with which it deals. In practice, however, software distribution can involve four or more levels. For example, an application software vendor might license its application program to a second software system vendor that would embed that applications program in an integrated software system; and the software system supplier might then license that system to a number of different computer hardware manufacturers, that would in turn license the software system with their hardware added to integrated systems manufacturers for distribution through distributors, international marketing companies and dealers. With so many parties involved, the complexity, cost and delay involved in negotiating and drafting satisfactory li-

96. See supra pp. 39-40.
97. It is the author's experience that administrative compliance with software licensing agreements is generally poor for three reasons. First, much software development and licensing occurs in the context of small entrepreneurial businesses that have neither the attitude nor the resources for careful monitoring of contract compliance. Secondly, most business people, not surprisingly, do not understand the subtleties and fine distinctions made by the laws respecting intellectual property in software. Thirdly, (see notes 17 & 18) copying of software is so easy and inexpensive that without strong internal mechanisms to prevent it, most businesses are unable to control the temptation of their technical employees to do what comes naturally in order to solve technical problems. As a result, copyright laws are often observed in the breach where software is concerned, especially in research and development laboratories where the chance of detection and prosecution is minimal.
cense agreements at all levels rapidly escalates.98

IV. LEGISLATIVE SOLUTIONS

A. The Problem

The crux of the difficulties with current software licenses is that the operative areas of copyright and trade secret law were developed to govern things that have nothing to do with software. Copyright law was originally developed to protect books99 although Congress has engrafted a number of provisions on it to deal with such modern devices as records and television.100 As commentators have observed101, however, copyright law was developed to protect communication between people, and not instructions intended for the operation of a machine.

As for trade secret law, it was not developed to protect secrets disseminated to the public in a form not readily understood by people that is intended to be used to operate machines. The result of this misdirected application is, as one might expect, that neither copyright law nor trade secret law precisely fits the software industry, and that both impose upon the industry undesirable uncertainty and administrative and legal expense.

B. Repeal of First-Sale Doctrine for Software

The first step in solving these problems should be a repeal of the first-sale doctrine of copyright law,102 at least as it may apply to

98. To take one example from the author's experience, a software vendor might license software to a computer hardware manufacturer that in turn sells hardware to a number of different computer systems manufacturers for incorporation into their products. To protect its trade secrets, the software vendor must attempt to control to some degree how the second-level manufacturers license the software to their end users. Usually, however, the first-level manufacturer does not know who the second-level manufacturers will be until after the master license with the software vendor is negotiated. Very often, the first level manufacturer must renegotiate the license to take account of needs of the second-level manufacturers that were not considered in negotiating the master license. Moreover, if one second-level manufacturer is able to negotiate concessions in the master license relating to the end-user license terms, other second-level manufacturers may see that second-level manufacturer's end-user license in the marketplace and may demand similar concessions. The prospects for renegotiation may seem interminable, and the transaction costs high, especially to a small software vendor dealing with large manufacturing clients.


100. See, e.g., 17 U.S.C. § 111 (secondary transmissions by cable systems and others); § 114 (phonorecords); § 116 (juke boxes) (1978); Goldstein v. California, 412 U.S. 560, 562 (1973).

101. See, e.g., CONTU Report, supra note 17, at 28-30 (Commissioner Hershey, dissenting).

102. See supra pp. 39-40.
computer software. While the doctrine conforms to established customs and human expectations where books are concerned, it is inappropriate for computer software for four reasons. First, computer software is not as portable as books. Each computer program is designed to operate on a specific computer with a specific operating system or, less commonly, on a specific range of computers with specific operating systems. A computer program intended for one machine may be useless on another or, worse yet, may appear useful but may produce disastrous latent or unexpected errors when used on the incorrect machine. The software vendor has a strong and legitimate interest in ensuring that this does not happen—its desire to ensure continued satisfaction of its customers. Secondly, custom and the vendee's expectations are different for books than for computer programs. A book buyer views a book as a piece of personal property like any other chattel. In contrast, a user of computer software is normally conscious of obtaining an intricate high-technology article that is to be used only to perform useful work under specifications and operating instructions provided by the vendor. Thirdly, and most importantly, even with the advent of inexpensive photostatic copying machines, the cost of copying a book is roughly comparable to the cost of purchasing it, so that the temptation for illicit copying is not too great. In contrast, the cost of copying computer programs is often several hundred to several thousand times less than the cost of obtaining legitimate licenses from the vendor, so that there is great temptation for borrowers to misuse the rental right subsumed under the "first-sale" doctrine. Finally, although some books require periodic supplementation or updating, the vast majority of books are complete in themselves. Unlike books, software usually requires periodic maintenance, support, correction and modification by the vendor to realize the functions that the customer desires.

103. For example, at a typical "consumer" cost of 5 cents per page, it would cost $20 to copy a 400-page book, roughly the cost of purchasing a book of that length in hardbound form.

104. The cost of copying consumer software such as VisiCalc or Lotus 1-2-3 is the cost of a floppy diskette, approximately $2, plus a few minutes of time. The license fee for legitimate use, however, may range from one to several hundred dollars. The disparity between the cost of illegitimate copying and licensing large programs for mainframe computers can be much greater.

105. Software, particularly if it is a new product, is seldom marketed without "bugs." Users protect themselves by demanding extended warranties if they have sufficient bargaining power, or by paying for maintenance service by the vendor, which can consist of efforts to fix "bugs" or the delivery of standard maintenance "updates," consisting of corrections to the software and perhaps new features.
For all these reasons, the first-sale doctrine should not apply to computer software. Indeed, if copyright law is to reflect the norms of actual use in the industry, it should restrict the end user’s rights to use of the software for the end user’s internal purposes on the computer system for which the software is obtained. This is in fact what most software vendors expect, and this is what most software licenses currently provide in one form or another. Of course, the current statutory provision for making and storing backup and archival copies of the software 106 should be retained, but it should be limited to the purposes for which the user’s copy of the software may be used.

Whether or not this precise formulation is enacted into law, the first-sale doctrine, as applied to computer software, should be changed. The proper purpose of any such doctrine should be to specify the norms governing private relationships in the absence of explicit agreement. If the doctrine is to have maximum usefulness, it should specify a legal relationship that is in accordance with predominant custom and the natural expectations of the parties. In the realm of software, this custom and expectation is definitely not that the software user has an unrestricted right to transfer, lease and lend the software. Producers of video cassette works, for similar reasons, have achieved some movement in Congress to repeal the first-sale doctrine as applied to video cassettes; 107 perhaps the software industry can piggyback on their efforts and achieve repeal of the doctrine for computer software as well.

C. Trade Secret Law

The problem of inappropriateness of trade secret law for computer software is not so easy to resolve for two reasons. First, although the Uniform Trade Secrets Act has achieved passage in several states, 108 trade secret law is generally not a creature of statute, but of the common law. Secondly, trade secret law is a creature of the individual states, not the federal government, so uniform enactment in all the fifty states would be necessary to achieve appropriate change, in the absence of federal legislation.

A solution can be approached, however, by considering the

objectives that a modified form of trade secret protection should satisfy. First, like existing trade secret law, any new form of software protection should resolve the uncertainty of copyright law by providing unambiguous coverage for the ideas, algorithms and other functional elements of computer software, and not merely their expression in a particular fashion in computer code. Secondly, like existing trade secret law, any new form of protection should not allow anyone to appropriate for himself what everybody knows, that is, what lies in the public domain. Thirdly, in order not to hamper the healthy and rapid growth of the software industry, any new form of protection should not protect against truly independent creation of the same idea or algorithm by another. Finally, in order to render the law certain in application, any new form of protection should not depend upon a lengthy or expensive government application process or an *ex post facto* assessment of the reasonableness of steps taken to secure practical protection by the software vendor.

A curious thing about this list of objectives is that, except for the first, all are satisfied by copyright law as it currently exists in the United States, but for the principle of Baker v. Selden. In that case, an author had developed a system of accounting forms now known as "T-accounts" and had written a book about them, and his heir sued the author of a similar book for copyright infringement. Since the other book had not copied the original author's expression, and the United States Supreme Court held that neither the idea of a system of T-accounts nor their form were proper subject matter for copyright. In this decision, the Supreme Court invoked the fundamental, if somewhat elusive, distinction between expression and idea as the touchstone of copyright protection, which is now enshrined in the Copyright Act at Section 102(b).

Were it not for this distinction, copyright law would provide an ideal sort of protection for computer software. Just as copying of expression may be proven today by evidence of access and substantial similarity of result, so could copying of ideas and algorithms from a computer program be demonstrated or disproved by focusing on access and substantial similarity of result. Nor would the law impede healthy and rapid development of the software industry, for anyone would have the right to use precisely the same

---

110. 101 U.S. 99 (1879)
111. See Wickham v. Knoxville Int'l Energy Exposition, 739 F.2d 1094 (6th Cir. July 12, 1984) (evidence of both access and substantial similarity required to prove copyright infringement).
idea or algorithm as another, as long as it had been taken from the public domain (i.e., general knowledge) or developed truly independently of the other's software effort. Thus, appropriate protection for computer software might be achieved through the simple expedient of repealing the principle of *Baker v. Selden* as embodied in Section 102(b) of the Copyright Act, for computer software only.

If confined to computer software only, such a step would not frustrate the policy objectives underlying the *Baker* principle. One such policy involves the overarching importance of maintaining the free flow of ideas in a free society. This flow is so important that its hindrance cannot be tolerated even if limited to a prohibition on copying the ideas of others. In this sense, the principle underlying *Baker v. Selden* has First Amendment overtones. Computer software, however, is not intended to communicate with people, but to control the operation of the machine. Accordingly, restriction on the copying of functional ideas used in controlling a machine, as opposed to their independent creation, would appear not to entail the dangers to free speech involved in a similar restriction on communications between people.

The second policy underlying the *Baker* decision was a desire to preserve the conceptual distinction between copyrights and patents. The *Baker* court ruled that statutory protection for ideas, such as the system of T-accounts for bookkeeping at issue there, should be protected, if at all, only by a patent grant. Since the *Baker* decision, however, the "nonobviousness" standard for patentability has developed through case law and revision of the patent statute into a formidable barrier to protection. As discussed above, this barrier is entirely too high to protect from piracy the many steps toward innovation, individually small but collectively significant, that generally occur in the course of developing a computer program. Moreover, patent protection, which prohibits independent creation of patented inventions, would dampen

---

112. The selective "repeal" of the *Baker* principle would break no rule of intellectual property legislation. Congress has established precedent for enacting special rules for special subject matter, particularly high-technology subject matter, in the existing Copyright Act. See supra pp. 52-53 and the Semiconductor Chip Protection Act of 1984, Title 3 of H.R. 6163, to be codified in chapter 9 of Title 17, United States Code.


114. Id. at 107.

115. The "obviousness" standard had its beginning in Hotchkiss v. Greenwood, 52 U.S. (11 How.) 248, 267 (1851), and was developed judicially for a hundred years before being enacted into the patent statute in 1952. See *Graham v. John Deere Co.*, 383 U.S. 1, 10-17 (1966).

116. See supra pp. 34-35.
innovation in the software industry. Thus the underlying concern of *Baker*, that utilitarian function should be protected, if at all, by patents, would appear inappropriate for software.

There appears to be nothing in the Constitution to prohibit the selective repeal of the *Baker* principle. Even for patents, the Constitution contains no specific requirement. The concept of "nonobviousness" is a creature of case law that ultimately became enshrined in the statute, and the criteria of novelty and usefulness are creations of the statute.\(^\text{117}\)

Alternatively, protection could be achieved under the copyright branch of the Copyright Clause, for there is nothing in the Constitution to prohibit applying copyright protection to ideas. This principle was established only by the *Baker* decision itself.

V. CONCLUSION

Last year, the Japanese Ministry of International Trade and Industry introduced into the Japanese Diet proposed legislation to provide a new form of protection for computer software.\(^\text{118}\) One reason given for proposing this legislation was that copyright law, designed as it is to protect literary works, was inappropriate for computer software. In Japan, there is no such thing as common-law trade secret protection, backed by injunctive relief, although similar protection may be achieved by contractual provisions.

The major objection to this proposed legislation was that the proposed term of protection would be too short.\(^\text{119}\) The proposal was viewed by some as a thinly-veiled attempt to allow the Japanese to gain the advantage of foreign software technology in a short period of time. Nevertheless, the proposal addressed a fundamental need, particularly in Japan, for legal protection of software more appropriate to the medium, function and business of the software industry.

While trade secret protection provides more secure coverage of software in the United States than in Japan, its genesis in the common law necessarily results in vagueness and uncertainty. Embodiment of common-law principles in the Uniform Trade Secrets Act will not substantially alleviate the vagueness and uncertainty of the common law because that statute generally follows the common


\(^{118}\) See PAT., TRADEMARK & COPYRIGHT J. (BNA) No. 669, at 424-25 (Mar. 1, 1984).

\(^{119}\) See id. The MITI proposal also would have required compulsory licensing in certain cases.
law.\textsuperscript{120} What is needed is a new form of protection appropriate to the unique features of computer software — an expression of the mind of man that serves the utilitarian function of controlling a machine. Perhaps selective repeal of the first-sale doctrine as applied to software and a reexamination of the longstanding principle of \textit{Baker v. Selden} in the context of software only would permit the law to achieve appropriate protection without impeding development in the software industry or requiring expensive, individualized negotiation over every business transaction involving computer software.

\textsuperscript{120} See generally Klitzke, \textit{supra} note 107.