ARTICLE

PROTECTING COMPUTER SOFTWARE: A COMPREHENSIVE ANALYSIS

Duncan M. Davidson

Contents

PROTECTING COMPUTER SOFTWARE: A COMPREHENSIVE ANALYSIS

Duncan M. Davidson*

Protecting computer software has become increasingly important. The market for software will have tremendous growth in this decade. Software is expensive to create but easy to copy. Proliferation of functionally identical computers has led to widespread software portability. Many software vendors, notably videogame distributors, are suffering from piracy and copying similar to that which significantly reduces the revenues of companies in the musical recording and videocassette businesses.

The debate over software protection has centered on the theoretical question of what is the nature of software. In the patent area, the issue has been whether software states mental steps or incorporates mathematical algorithms or other fundamental laws of nature which traditionally are not patentable. In the copyright area, the issue has been whether the form of software which is actually used on the computer, binary form, is in any sense a "literary work" since it is intended only to operate a machine.

What causes difficulties in the analysis of software protection is that software is both *symbolic* and *mechanical.* Software can be directly used to operate a machine, but it only symbolically represents the hard-wiring of the machine. Software creators no longer need to understand the actual wiring and circuitry of the computer; it is sufficient if they understand how to manipulate certain "computer languages" around which other engineers, who are knowledgeable of hardware, have built the computer. Instead of building software, software engineers write it. Because of this, software has been argued to be suitable for

*Duncan M. Davidson has recently left Los Angeles to practice law in Denver, Colorado. He is the Chairman, Committee on Proprietary Protection of Software, Computer Law Division, American Bar Association Section of Science and Technology. The views expressed are personal to the author and do not necessarily represent the views of the American Bar Association or any other organization. The author would like to acknowledge the following people, who, while not necessarily agreeing with the author's views, provided valuable assistance in the ideas expressed in this article: John Lautsch, Chairman, Computer Law Division; Jean Kunkel, Los Angeles; Richard L. Bernacchi, Los Angeles; Mary Carrington, Newport Beach, CA; G. Gervaise Davis III, Monterey, CA; Hugh D. Finley, San Francisco; Robert T. Burke, San Francisco; Scott Price, Washington, DC and Ronald J. Palenski, ADAPSO, Arlington, VA.

copyright protection. Software is used also to manipulate computer machinery, however, making it part of a machine or process. Because of this, software has also been argued to be suitable for patent protection. The dual nature of software is the basis of the theoretical debate which has caused much concern and consideration.

This article first analyzes the nature of software and concludes that for most purposes it should be thought of as a tangible engineering product. The article next reviews patent, copyright, trade secret, international and technological protective schemes and concludes that patent, copyright and trade secret law together can adequately protect software once its nature is fully understood. Patent protection covers software when it is used in larger processes or as part of machinery in a manner which dynamically and automatically affects items in the surrounding environment. Copyright law protects the computer program itself—the specific logic and design of the software. Trade secret law protects the program's underlying algorithms and the way in which the program interfaces with other components of a process or machine.

Finally, the article discusses proposed new protective schemes, including current proposed amendments to the 1976 Copyright Act. Certain changes to these amendments are recommended in order to avoid needless litigation and to promote, rather than hamper, the rapidly growing software industry.

THE NATURE OF SOFTWARE

It may be possible to work out a satisfactory scheme of software protection by a functional approach, considering the purposes being served by protection, without answering the formal question of what is the nature of software. In order to appreciate and correctly apply this functional approach, however, it is still necessary to understand what software is and what it is not.

1. Definitions

Because of the various meanings given to *software,*[1] the following terms will be used in this article. *Software* shall refer to all materials encompassing or

[1]Usually, *hardware* refers to the computer machinery and *software* refers to the machine-readable programs used to operate the hardware. *See* Law Research Serv. v. General Automation, Inc., 494 F.2d 202, 204 n.3 (2d Cir. 1974) ("*software* includes the punch cards, memory tapes, and paper tapes programmed to instruct the computer"); Com-Share, Inc. v. Computer Complex, Inc., 338 F. Supp. 1229, 1231 (E. D. Mich. 1971), *aff'd per curiam* 485 F.2d 1341 (6th Cir. 1972) ("*software* refers to the programs and controls which are used in the computer").

Software often is given a more expansive meaning, to include both "human-readable" versions of programs, related documentation and operating manuals, University Computing Co. v. Lykes-Youngstown Corp., 504 F.2d 518, 527 (5th Cir. 1974); Accountants Computer Serv., Inc. v. Kosydar, 35 Ohio St. 2d 120, 123-24, 298 N.E.2d 519, 522 (1973), and "support" services such as advice, programming systems, and engineering help, Honeywell, Inc. v. Lithonia Lighting, Inc., 317 F. Supp. 406, 408 (N.D. Ga. 1970) (defines *hardware* and *software* and complains of the lack of clarity in the "jargon" used in the computer industry).

describing computer programs. *Computer programs* are the ordered set of instructions which can operate a computer.

Software is first begun by creating "source" materials, including program descriptions like flow charts and functional specifications. These source materials are then used to create the program. These source materials are used to draft computer instructions in a specific computer language. These written instructions are the "source code" of the program. Source code can be written in languages which are English-like, such as BASIC or FORTRAN. Interspersed between the source code instructions can be extensive comments and annotations in English which describe the various computer instructions and which enable an untutored programmer to understand the program. Source code can also be written in wholly symbolic languages, such as APL or direct machine language (ones and zeros), without an English language overlay.

Source code instructions are either directly used by a computer or are first translated into the computer's machine language as "object" code. Object code usually is printed as ones and zeros, but it can also be printed as octal numbers (0-7) or hexadecimal numbers (0-15, with A-F representing decimal 10-15). Object code can be directly translated into "assembly" language, in which machine instructions are represented by mnemonics. For example, if "10110010" represents the machine instruction "Add register B to whatever is in the accumulator," an assembly language instruction might read "ADD B." (The same instruction in octal numbers would be "262" and in hexadecimal numbers would be "B2.")

The written code (source or object) is prepared in machine-readable "binary" form, the form the computer uses. Binary form is unintelligible to a human being and normally resides in computer memory or on magnetic media in the form of binary electrical signals or magnetization. Binary form can be represented by ones and zeros, but its actual form is slightly different; electrically, it usually consists of high and low voltages, and magnetically, it consists of different polarities of magnetization.

Binary form and object code are often confused. The binary version of a program is sometimes called the object version. While binary form can always be printed out into ones and zeros, these ones and zeros do not necessarily represent the object code of a program. Source code, and other textual material (such as on a word processing machine) are stored in a standard encoded form in a computer, such as in accordance with the encoding protocol "ASCII." The capital letter *K* in ASCII, for example, is encoded as binary 01001011, which is decimal 75, hexadecimal 4B, and octal 113. To print textual material, the computer would be instructed as to the encoding protocol (ASCII or otherwise) and would translate each binary number to a letter, number or punctuation.

Object code, the direct symbolic representation of the machine language, is intelligible to trained engineers. Indeed, it has no other function when printed but to be painstakingly understood. At one time, object code was source code, the symbolic language in which programmers wrote. Today, they write in "higher-level" languages such as BASIC. A program or several programs in combination, called "compilers" or "assemblers," depending upon the nature of the computer language, transform higher-level source code into object code in binary form which is used by the computer for operation. In some computer languages, the higher-level source code encoded in binary form is directly translated into computer operation by programs called "interpreters."

Software consists of source materials, source code, object code, and binary form.

2. Common Misconceptions

a. *Engineering, Not Art*

A major fallacy in the literature of software protection has been to view software as the result of artistic or scientific creativity. There certainly is creativity in the construction of software, but the creativity is more like that involved in any engineering creation. Software is created like most engineering products: by a process of problem or project definition, followed by designing the product (the program), creating a prototype (writing the source code), testing the prototype (debugging), and ultimately realizing a commercially marketable product.

The fact that source code looks like a writing is misleading. A program's sentences and paragraphs must be very precisely drafted according to how well it can be "read" by a machine, not by a human. In the earliest days of the industry, programs were hard-wired into machines, a process which was painstaking and time-consuming. The great development in computing which has led to its wide proliferation was the concept of symbolic software, written in symbolic languages, which could substitute for the hard-wiring. A hard-wired machine can still perform the same functions much faster than a general purpose programmable machine, but the time to develop and test a general purpose computer's symbolic program is much less than that required for a hard-wired machine. The development of programming languages enables the relatively fast creation of "new" machines, general purpose computers which have been uniquely programmed, and these machines can be developed by software engineers who have no substantial knowledge of how the machine works or is wired.

The failure to recognize the engineering and mechanical nature of software has caused fundamental problems in the protection of software. For example, it has colored the manner in which patent claims and patent issues have been argued, because by divorcing programming from the hard-wiring of machines, courts and others have been led to the conclusion that programs are not part of the machinery but are "mental" or "mathematical" processes.

Another example is the superficial attractiveness of *sui generis* protective schemes based upon a "design patent" type of law. In the United Kingdom, for example, a device made from a blueprint is considered a copy of a blueprint.[2] A similar type of law could apply to all forms of software, including object code, source code, flow charts and other representations of the same fundamental design. Design patent laws, however, protect *visual* designs, not engineering architecture. Such a law for software, because of software's close relation to hard-wiring, could restrict reverse engineering of otherwise unprotectable computer design and architecture. This would cause greater anticompetitive effects than current protection. A new design patent-like law for software would require careful consideration.

A third example of the problems created by this fallacy is the continuing debate on the copyrightability of software. The fundamental question of copyright application to software, discussed below, is whether software used solely for utilitarian purposes when in a machine should be copyrightable. Software is not only utilitarian; it is also symbolic. The symbolic nature of software distinguishes it from other utilitarian objects and from "designs" and, as will be discussed, makes copyright protection appropriate. This issue has been misunderstood and misstated because the engineering nature of software is overlooked in certain forms of software and found in other forms. It is in all forms. The issue cannot be finessed by making artificial distinctions.

Nevertheless, the prevalence of this fallacy indicates that software engineering is a new type of engineering and requires a reconsideration of the policies and theories underlying both patent and copyright protection.

b. *Tangible, Not Intangible or Unique*

A major debate has been over whether software is tangible or intangible, or a new, hybrid creation somehow existing somewhere between tangible and intangible objects. This debate has led to such oxymoronic decisions as saying that software "though intangible" is nevertheless a "good" under the Uniform Commercial Code (a "good" being defined as a movable *thing*).[3]

Software has tangible and intangible elements.[4] The tangible elements include written or printed copies and all machine-readable binary forms of object and source code, which of necessity must be tangible to be "read" by a ma-

[2]L.B. Plastics Ltd. v. Suish Prod. Ltd., [1979] Fleet St. Rep. ___.

[3]Triangle Underwriters, Inc. v. Honeywell, Inc., 457 F. Supp 765 (E.D.N.Y. 1978), *rev'd on other grounds*, 604 F.2d 737 (2d Cir. 1979).

[4]The difference between the tangible and intangible parts of software can be analogized to the difference between the tangible and intangible parts of music. These can be expressed in the following chart:

Music	**Software**
Intangible Elements	*Intangible Elements*
type of music; musical techniques; melody; orchestration; "hooks"; performance.	functionality; programming techniques; realization; algorithm; reduction to practice.
Tangible Elements	*Tangible Elements*
description; conductor's manual; sheet music; score; record/cassette tape.	description; flow charts; source code; binary form.

chine. The intangible elements include a program's external attributes, such as what functions it performs, and its internal design, which includes the underlying concepts, ideas and programming techniques and the specific algorithms using the underlying programming techniques.

With strong tangible and intangible elements, it can be difficult to decide whether ''software'' is the intangible set of instructions or a tangible embodiment of the instructions. This debate, however, relates more to the definition of *software* than to the underlying nature of software. Any engineering creation has intangible elements, but usually the intangible ideas upon which the creation functions are not confused with copies of the creation.

For most purposes, like other engineering products, *software* should be considered tangible. The *program* should be considered intangible; it is the specific design and logic of the software. It is the physical software, however, which is the subject of most legal transactions concerning any program. The physical software can operate a computer. It is therefore unlike other intangibles. It is unlike intangible legal rights; a patent right does not operate a patent. It is unlike intangible laws and principles; the law of physics do not create patentable machines. It is also unlike specific, ordered intangible instructions; a set of rules, such as how to write a computer language or how to implement ''protocol'' for two computers to talk to each other, does not by itself create computer programs or talk to computers.

It is critically important to appreciate this distinction between software and the program when analyzing legal transactions. Software is too easily defined as a hybrid tangible/intangible object or an amorphous item coexisting both in objects and as the intangible processes accomplished by the program instructions. This is reflected in a statement that software ''though intangible'' is a ''thing.'' Software is viewed like the disembodied grin on the Cheshire cat in *Alice's Adventures in Wonderland:* software is not seen as the cat itself (a copy of the software) nor the unique pattern of stripes which identifies the cat (the specific design and logic of the program); software is viewed as in between, like the grin—not as tangible as the cat but not intangible either, since software can operate a computer.

The grin should be removed from the face of this Cheshire cat. The ''grin'' fallacy arises because of the common psychological errors of extrapolating concepts into real things or abstracting things into concepts. The nature of software exacerbates these tendencies. First, software is easy to copy. Thus, it can be argued that a ''thing'' exists separate from any one copy (but found in all copies) which flows from copy to copy as the copies are made. But that ''thing'' is only the fact of resemblance, which is not a thing at all (although it may be protectable under laws such as the Copyright Act, which gives rights to preclude others from creating a substantially similar ''pattern of stripes''). Second, the same functioning of a computer program can be achieved with different types of instructions. Again, it can be argued that software consists of this functioning and is a ''thing'' which exists separate from any particular form in which it may be expressed. But this specific functioning is no more than the particular expression of the intangible program. Different software can embody different programs that accomplish similar functions. Third, software can be electrically and electromagnetically transmitted. While transmitted, its existence is ephemeral, but not intangible. Machines can read it.

It would be better to view software as tangible than to continue the debate and define it as tangible for some purposes and intangible for others. It is too easy to slip into the grin fallacy. It is the *tangible* software which is being marketed; it is its *intangible* program which is protected by copyright or whose functioning which might be protected by a patent.

c. *The Architect's Drawing Fallacy*

Analogies can be helpful, but they should be viewed initially with skepticism. A more traditional analogy for software (other than to a Cheshire cat) is to view the dichotomy between source code and binary form as analogous to the difference between blueprints and devices, recipes and cakes, and architects' drawings and buildings.[5] These analogies are only superficially accurate.

The devices, cakes and buildings are mechanical in nature, like binary form, but are not easily copied, and do not retain symbolic attributes such as source or object code which can be reproduced from binary form. The blueprints, recipes and architects' drawings are symbolic, like source or object code, but cannot be mechanically translated into devices, cakes or buildings without human labor. The three analogies lack the automatic, mechanical translation of source or object code into binary form and then into machine operation which makes software both symbolic and mechanical. Therefore, these three analogies are inapposite.

d. *Courts in Wonderland*

These fallacies and others have been carried over into judicial and regulatory decisions. In the early cases, software was characterized as being "knowledge."[6] Perhaps those courts were anthropomorphizing computers as "knowing" what they have in their memory systems.

There has been a continuing vigorous fight to avoid sales and use taxes on software. The parties leading the fight have characterized software as "intangible" because most of these tax laws apply only to "tangible personal property."[7] Under certain tax situations, software has been found to be tangible.[8]

[5]*E.g.*, Stern, *Another Look at Copyright Protection of Software*, 3 COMPUTER/L. J. 1, 7 n.23, 10 (1981).

[6]*E.g.*, District of Columbia v. Universal Computer Assoc., 465 F.2d 615, 618 (D.C. Cir. 1972).

[7]The following cases (not exhaustive) held that for purposes of a personal property or a sales or use tax, software (at least in certain forms) is an "intangible." District of Columbia v. Universal Computer Assoc., Inc., 465 F.2d 615 (D.C. Cir. 1972) (punched cards containing programs); County of Sacramento v. Assessment Appeals B., 32 Cal. App. 3d 654, 671, 108 Cal. Rptr. 434, 446 (1973); Commerce Union Bank v. Tidwell, 538 S.W.2d 405 (Tenn. 1976) (that software was on magnetic tapes and punched cards not determinative; software could have been telecommunicated and therefore is intangible); Honeywell Information Systems v. Maricopa County, 118 Ariz.

Distinctions are now being made, such as between custom designed software and packaged software, the former being considered inherently a transaction for services with only incidental and unimportant tangible elements.[9] Nevertheless, many states tax some types of software and many states exempt all types, with little consistency.[10]

The tangibility issue has also arisen under the Uniform Commercial Code, where software is generally found to be a "good" and covered.[11] In isolated decisions in other areas, software has been found to be sufficiently tangible to

---

171, 575 P.2d 801 (1977) (software sold as part of integrated system must be unbundled for purposes of exempting it from personal property taxation); State of Alabama v. Central Computer Serv., Inc., 349 So. 2d 1160 (Ala. 1977) (held, that although there is an incidental commingling of intangible information with tangible magnetic tapes and punched cards in the sale of computer programs, the essence of a software transaction is the purchase of nontaxable intangible information; the Alabama court reached this conclusion despite a prior holding that the leasing of a motion picture film to a local TV station was taxable because the film was tangible personal property); First Nat'l Bank v. Bullock, 584 S.W.2d 598 (Tex. 1979); First Nat'l Bank v. Department of Rev., 85 Ill. 84, 421 N.E.2d 175 (1981) (held that the substance of a software transaction is not the tapes but the "information"); James v. TRES Computer Serv., Inc., 642 S.W.2d 347 (Mo. 1982) ($135,000 program for on-line customer information system transferred under license on a $50 magnetic tape *held* intangible technical professional services in part because it *could* have been telecommunicated). *See also* Spencer Gifts, Inc. v. Taxation Div. Dir., 182 N.J. Super. 179, 440 A.2d 104 (1981) (lease of magnetic tapes containing mailing lists nontaxable); Janesville Data Center, Inc. v. Wisc. Dept. of Rev., 84 Wis. 2d 341, 267 N.W.2d 656 (1978) (sale of magnetic media-containing customer data nontaxable); Bullock v. Statistical Tabulating Corp., 549 S.W.2d 166 (Tex. 1977) (sale of media-containing data supplied by customer nontaxable); Central Data Sys. v. Kosydar, 35 Ohio St. 2d 120, 298 N.E.2d 519 (1973) (sale of reports containing business analysis nontaxable). *See generally* Note, *The Nature of Taxability of Computer Software,* 22 WASHBURN L.J. 103 (1982).

[8]The following cases (not exhaustive) held that software was tangible under various tax statutes. Greyhound Computer Corp. v. State Dept. of Assessment & Taxation, 271 Md. 674, 320 A.2d 52 (1974) (software is tangible personal property and is subject to the personal property tax); Texas Instruments v. U.S., 407 F. Supp. 1326 (N.D.Tex. 1976), *rev'd,* 551 F.2d 599 (5th Cir. 1977) (in line with the IRS,VB42's long-standing position that software is intangible, the District Court denied a claim for an Investment Tax Credit under IRC § 48, which is only available for "tangible personal property," for certain software, because "the real investment is in the information on the tapes which is intangible." The Fifth Circuit reversed and held that property is intangible if its intrinsic value is attributable to its intangible elements rather than to any of its specific embodiments. The court analogized software to motion picture and TV films, which are tangible for Investment Tax Credit purposes, by noting that software physically embodies the information on the tapes in much the same manner as performances are captured on film).

[9]*See* California Dept. of Equalization Regulation 1502 (1972); Bill 2932 and CALIF. REV. & TAX. CODE § 6010.9 (September 22, 1982) (finding "custom computer program" creation a service not subject to sales or use tax).

[10]A recent count showed thirty-four states tax at least some types of software and sixteen states and the District of Columbia exempt it, although sales of some items containing software even in those states may be taxable. 1980 ADAPSO TAX SURVEY (updated Feb. 17, 1983) (listing as exempting it: Ala., Ariz., Colo., D.C., Fla., Ill., Ind., La., Md., Minn., Neb., N.J., N.Y., No. Car., No. Dak., Tex. and Vt.). *See generally* BIGELOW & SALTZBERG, STATE COMPUTER TAX REPORT (1982).

[11]*See* Note, *Computer Programs as Goods under the U.C.C.*, 77 MICH. L. REV. 1149 (1979). When software is licensed incidental to hardware procurement, it is treated as a good since the whole transaction is treated under the UCC. *E.g.*, Investors Premium Corp. v. Burroughs Corp., 389 F. Supp. 39 (D.S.C. 1974); Carl Beasley Ford, Inc. v. Burroughs Corp., 361 F. Supp. 325 (E.D. Pa. 1973), *aff'd,* 493 F.2d 1400 (3d Cir. 1974); Burroughs Corp. v. Joseph Uram Jewelers, Inc., 305 So. 2d 215 (Fla. Dist. Ct. App. 1974); Bakal v. Burroughs Corp., 74 Misc. 2d 202, 343

be subject to replevin,[12] to be a product which can be the subject of an illegal tie-in violating of antitrust laws,[13] and to be property subject to conversion.[14] In the Commerce Department's Export Regulations, software can be both intangible "technical data" and a tangible "controlled commodity"; the Department has taken the position that software is technical data, although in some material forms it will be embodied in a controlled commodity.[15]

It is likely that over time the courts will shift from the original view that software is intangible or "knowledge" to a view that software is tangible, at least in certain types of transactions.[16] This will occur largely when software is seen to be more like other tangible items, as it can be in computer stores where software is sold like record albums. This will also occur when attorneys and judges consider the practical or functional aspects of software transactions. For example, in transactions for acquiring software, software can be acquired as is, perhaps with some customization, or can be newly developed or largely specially manufactured. If software is largely specially manufactured, the software contract should include many of the elements of a construction or service contract. It can be strongly argued that the resulting tangible software is incidental to a larger service arrangement and, therefore, the transaction as a whole is for an intangible (programming services). In most cases for acquiring software, the software is either sold as is ("packaged") or is only moderately customized from its packaged form. Such software transactions on a functional level should be handled under contracts which are analogous to sales or lease contracts for tangible products.

3. Conclusion

Software is a new type of technology which will become dominant in the coming years. This technology has at least two novel aspects: it is expensive to

N.Y.S.2d 541 (1972). When it is thought of as the "grin," software, "though intangible," has been held to be a good. Triangle Underwriters, Inc. v. Honeywell, Inc., 457 F. Supp. 765 (E.D.N.Y. 1978), *rev'd on other grounds*, 604 F.2d 737 (2d Cir. 1979). In holding that computer programs are goods under the UCC, the District Court stated: "Although the ideas or concepts involved in the custom designed software remained Honeywell's intellectual property, Triangle was purchasing the product of these concepts. That product required efforts to produce, but it was a product nevertheless and, though intangible, is more readily characterized as 'goods' than 'services'."

[12]F & M Schaefer Corp. v. Electronic Data Sys. Corp., 430 F. Supp. 988 (S.D.N.Y. 1977) EDS sought replevin of software from F&M, and F&M resisted the replevin by claiming that software consists of services as well as intengible ideas and concepts. The court held that software is tangible and that EDS could replevin the software by recovering from F&M the tapes, instructions, supporting documents, and all copies of each of the above.)

[13]*In re* Data General Corp. 490 F. Supp. 1089 (N.D. Cal. 1980) (operating system software is a product which can be illegally tied), *appeal pending*.

[14]National Sur. Corp. v. Applied Sys., 418 So. 2d 847 (Ala. 1982).

[15]*Cp.* 15 C.F.R. § 379.1 and the Department of Commerce Controlled Commodity List, Commodity No. 1572A.

[16]Thus, most software transactions involving packaged or moderately customized standard software should be subject to sales and use taxes, the investment tax credit and accelerated cost recovery under the Internal Revenue Code, the Uniform Commercial Code (assuming the transaction is analogous to a sale or long-term lease), replevin, and so on.

make but easy and inexpensive to copy; and it can operate a machine but can be written symbolically, without any awareness of how the machine is built. This technology represents a major advance in the art of the design of machines.

The nature of software is certainly complicated, but it should not be misunderstood. Despite its novel aspects, it is fundamentally analogous to other engineering creations. In physical form it can vary from a mechanical element which can operate a machine (binary form) to symbolic engineering representations of the hard-wiring of a machine (the source materials). It is tangible, but has basic intangible elements which can be "copied" either by copying the internal design of the program to accomplish the same functions, or by copying the functions with a different type of internal design.

The fundamental issue in the protection of software revolves around the latter two elements: is it good policy to protect only the internal design or is it better policy to protect the functions? This question has usually been posed as whether a copyright or a patent scheme is most appropriate.

PATENT

The need for protection of software was first felt by computer users and independent software houses in the early 1960s. Many turned to patent protection and soon the Patent Office was flooded with applications.[17] In 1965, a presidential commission studied the issue and recommended against patent protection.[18] For fifteen years thereafter, patent protection of software was rare and uncertain.

1. The Subject Matter Tests

a. *Mental Steps*

In 1966, the Patent Office issued guidelines for the patenting of software: software was patentable subject matter under 35 U.S.C. Section 101 if it consisted of "utility steps" and not "mental steps."[19] The Patent Office was applying the fairly recent decision of *In re Abrams*,[20] which explained what is now known as the mental steps doctrine: processes which are executable mentally are not patentable. This doctrine creates an obvious problem for patenting of

[17]Computer Software Protection: A Pragmatic Approach, Proceedings of the 1981 Fall Computer Law Association Meeting in Washington, D.C., at 37 [hereinafter referred to as CLA Transcript].

[18]The Commission was established pursuant to Executive Order No. 11215 (April 8, 1965); 30 Fed. Reg. 4661; 814 Off. Gaz. Pat. Off. 3. In 1966 the Commission determined that computer programs should not be patentable and recommended legislation to that effect. S. 1042, H.R. 5924. In 1967, legislation was proposed as § 106 to the 1952 Patent Code, holding that all computer programs are not patentable. 836 Off. Gaz. Pat. Off. 1115, 1118. The legislation was controversial and was promptly withdrawn. *See Commissioner Brenner's Testimony before the House*, H.R. 5924 and H.R. 13951, Feb. 28, 1968.

[19]829 Off. Gaz. Pat. Off. 1 (1966).

[20]188 F.2d 165 (CCPA 1951).

software because most computer software simply executes faster "mental" calculations or bookkeeping entries by machine than that which could be done manually.

The Patent Office was suddenly caught in a crossfire between the interests of the major computer manufacturers and the scientists and engineers who were then academically oriented, and the independent computer users and software developers.[21] The major computer manufacturers, such as IBM and Sperry-Univac, realized that widely available software helped sell their hardware. The software engineers considered themselves scientists, not entrepreneurs, and were more interested in advancing the art than in exploiting the business. Opposed to them were major oil companies and other users of computers who had internally developed extensive and complicated programs, and independent software houses which insisted that software is technology, not "science" or "art," which needed protection.[22] In 1968, the 1966 guidelines were withdrawn and replaced by narrower guidelines.[23] In 1969, the Patent Office was pressured into rescinding its guidelines.[24] The issue of the patentability of software was left either for Congress or the courts.

b. *Technological Arts*

The courts were first to take up this challenge. Beginning in 1969 and throughout the next decade, the Court of Customs and Patent Appeals (CCPA) was confronted by appeals from rejection of patent claims by the patent examiners and the Board of Patent Appeals for various claims relating to software. These claims arose either as inventions for new machines (apparatus claims) or for new methods for manipulating materials (process claims).

The CCPA first considered the mental steps doctrine. It allowed apparatus claims involving software on the grounds that the steps were performed by a machine and therefore were not "mental."[25] The CCPA then allowed process claims by creating a broad exception to the mental steps doctrine: method or process claims which could be performed on a programmable computer or which were otherwise in the "technological arts" were patentable subject matter because they were not "purely mental."[26]

[21]Nimtz, *Development of the Law of Computer Software,* 3 Comp. L. Serv. § 4-1, art. 6 (1979).

[22]The major oil companies have pursued a number of software patents, generally relating to seismic detection of subterranean oil. *E.g., In re* Musgrave, 431 F.2d 882 (CCPA 1970); *In re* Christensen, 478 F.2d 1392 (CCPA 1973); Parker v. Flook, 437 U.S. 584 (1978); *In re* Johnson, 589 F.2d 1070 (CCPA 1979); *In re* Sherwood, 613 F.2d 809 (CCPA 1980), *cert. denied,* 450 U.S. 994 (1981); *In re* Walter, 618 F.2d 758 (CCPA 1980); *In re* Taner, 681 F.2d 787 (CCPA, Jun. 10, 1982).

[23]855 Off. Gaz. Pat. Off. 829; 33 Fed. Reg. 15609 (Oct. 17, 1968).

[24]868 Off. Gaz. Pat. Off. 349; 34 Fed. Reg. 15724 (Oct. 3, 1969).

[25]*In re* Prater, 415 F.2d 1393 (CCPA 1969) (dictum); *In re* Bernhart, F.2d 1395 (CCPA 1969).

[26]*In re* Musgrave, 431 F.2d 882 (CCPA 1970) (method claims for computing seismic information from a variety of mathematical data). The "technological arts" test was reaffirmed in *In re* Foster, 438 F. 2d 1011 (CCPA 1971). *See In re* Mahoney, 421 F.2d 742 (CCPA 1970) (method for synchronizing receipt of digital information); *In re* McIlroy, 442 F.2d 1397 (CCPA 1971).

In *Gottschalk v. Benson,*[27] the U.S. Supreme Court curtailed the CCPA's expansionist exuberance and recharacterized the debate. It held that a patent claim relating to a simple mathematical formula without substantial practical application except in connection with digital computers was not patentable subject matter. The claim in question related to a method for converting one type of internal representation of numbers to another. The theoretical ground for this decision was that principles of science, like mathematical formulas, are not appropriate for patent protection. The Court also emphasized a functional reason for why this was so: granting patents in such cases could effectively preempt the use of the mathematical algorithm involved, giving the patent overly broad applicability. The Court expressly restricted its holding to the particular facts of the case to avoid stating a general rule. *Benson* was later interpreted narrowly in *Dann v. Johnston,*[28] the second Supreme Court decision considering the issue of patentability of software, in which the subject matter issue was not reached.

c. *Point of Novelty*

The *Benson* Court strongly urged Congress to act in order to clarify the area of patenting of computer programs. Congress did not act, but the CCPA searched for a new approach. The CCPA was careful to consider claims in light of *Benson.* It did this by applying what later evolved into the "point of novelty" test, a test which it had earlier rejected:[29] if the novel element in the claim was the use of software to implement a mathematical algorithm, the claim was nonstatutory subject matter. For example, in deciding *In re Waldbaum*[30] prior to *Benson,* the CCPA had allowed certain claims relating to a computer program process which was useful in counting the busy and idle lines of a telephone switch. On reconsideration in 1977, many of the claims were disallowed as being too abstract and sweeping because they were not tied to a particular application. Many claims which were tied to particular applications were also disallowed since their novel elements were algorithms and they would wholly preempt the algorithms for those uses.

When it could, the CCPA avoided the subject matter issue altogether.[31] In general, however, the CCPA recharacterized software claims as not involving software by itself but involving either particular programmed machines[32] or

[27]409 U.S. 63 (1972).

[28]425 U.S. 219 (1976).

[29]*In re* Bernhault, 417 F.2d 1395, 1399 (CCPA 1969).

[30]457 F.2d 997 (1972), *rev'd on reconsideration,* 559 F.2d 611 (CCPA 1977). *See In re* De Castelet, 562 F.2d 1236 (CCPA 1977) (method of generating curves which guide drafting and milling machines); *In re* Richman, 563 F.2d 1026 (CCPA 1977) (method for calculating an airborne radar correction angle); *In re* Christensen, 478 F.2d 1392 (CCPA 1973) (method of determining and plotting subsurface porosity).

[31]*In re* Knowlton, 481 F.2d 1357 (CCPA 1973) (affirmed rejection of claims relating to computer processing of list information in view of prior art); *In re* Comstock, 481 F.2d 905 (CCPA 1973) (reversed rejection of claims relating to electronic calculator because of lack of particularization).

[32]*In re* Noll, 545 F.2d 141 (CCPA 1976) (apparatus claims).

methods of operating or controlling machines more efficiently.[33] In essence, the CCPA began reconstruing software claims as being claims for a larger apparatus or process in which software was only part of the invention.

In *Parker v. Flook,*[34] the U.S. Supreme Court again deflated these expansionist tendencies. It reversed the CCPA, which had found a particular program to be a part of a larger process, because the only novel element in the process was a program implementing a mathematical algorithm. The Court said that recitation of insignificant postsolution activity would not make an unpatentable algorithm claim patentable. In other words, if the program only output the results of a calculation, tying the results to a specific end use was not enough to make the process as a whole patentable subject matter. This analysis broadened the "preemption" approach in *Benson,* for the patent in *Flook* would not have wholly preempted use of the mathematical algorithm involved. In effect, *Flook* made it extremely difficult to patent processes implemented by software.

d. *Freeman-Walter Two-Step*

Just before *Flook,* the CCPA handed down a key decision in this area. In *In re Freeman,*[35] the CCPA upheld claims relating to computerized control of typesetting based upon a two-step test:

> First, it must be determined whether the claim directly or indirectly recites a mathematical algorithm.
>
> Second, the claim must be further analyzed to ascertain whether in its entirety it preempts that algorithm.

This test is a clear exposition of a narrow interpretation of *Benson.* After *Flook,* this test was broadened and clarified in *In re Walter.*[36] The second step in the test was restated from one of preemption to the following:

> If the mathematical algorithm is implemented in a specific manner to define structural relationships between the physical elements of the claim (in apparatus claims) or to refine or limit the claim's steps (in process claims), the claim is statutory subject matter.
>
> If, however, the mathematical algorithm is merely presented and solved by the claimed invention and is not applied in any manner to the physical elements or process steps, no amount of postsolution activity nor limited field of use will render the claim statutory.

In *Diamond v. Diehr,* [37] the Supreme Court endorsed this approach. The Court upheld a claim containing a mathematical algorithm as a step in a process which as a whole performed patentable subject matter functions. The mere use of the program did not render the complete process unpatentable. The process

[33]*In re* Chatfield, 545 F.2d 152 (CCPA 1976), *cert. denied,* 434 U.S. 875 (1977) (process claims); *In re* Deutsch, 553 F.2d 689 (CCPA 1977) (method to control a system of multiple-unit oil refineries or manufacturing facilities); *See In re* Tomy, 575 F.2d 872 (CCPA 1978) (method of translating languages by computer).

[34]437 U.S. 584 (1978).

[35]573 F.2d 1237 (CCPA 1978) (computerized typesetting).

[36]618 F.2d 758 (CCPA 1980).

[37]450 U.S. 175 (1981).

in question was using a computer program to time the curing process for rubber. In what is considered a companion case, *Diamond v. Bradley,*[38] the Court by a split decision upheld a CCPA decision in which software designed to access certain otherwise unaccessible "scratch pad" registers in the central processing unit of certain computers was considered patentable subject matter.

Diehr does not explicitly open the door for the patenting of computer programs. Instead, the case established that in at least certain limited instances, computer programs are patentable when they are part of a larger process or apparatus which is patentable. In other words, a patentable process or apparatus which contains as one of its elements computer program calculations is not thereby rendered unpatentable.

Diehr again recharacterizes the debate and calls into question the point of novelty test and the holdings in both *Flook* and *Benson.* The patent claims allowed in *Diehr* are quite similar to those disallowed in *Flook.* In *Flook,* the computer program was used to monitor and time the distillation process for petroleum; in *Diehr,* the computer program was used to monitor, time and interrupt the curing process for rubber. The cases were distinguished on the grounds of novelty. In *Diehr,* the rubber-curing process used the programs in the measuring and input of information, the calculation and, if the timing were right, the curtailing of the curing process, all new elements in the art. In *Flook,* the only new element in the petroleum distillation process was the use of the program in calculating timing.

On the facts of the two cases, the basic distinction is that in *Flook* the program merely calculated and a person had to interrupt the distillation process; in *Diehr* the program calculated and interrupted the rubber-curing process without human intervention. This distinction may not be convincing, and to some it signals an encouraging change in the Supreme Court's views as to patentability of programs.[39] To others, it reveals a more fundamental disagreement among members of the Court over what was meant in *Benson* by a "mathematical algorithm."[40] The *Diehr* majority interpreted "algorithm" narrowly to mean a mathematical calculation or mathematical formula. The minority considered *algorithm* to be synonymous with *computer program,* the specific logic and design of the software. The solution chosen by the majority for defining algorithm is consistent with the approach taken by the CCPA: defining *algorithm* narrowly to increase the number of programs which would be patentable.

The CCPA has considered software-related claims in light of *Diehr.* In one case, certain claims relating to computerized axial tomography (CAT scans) were allowed, and others were disallowed.[41] One claim which was disallowed

[38]450 U.S. 381 (1981), *aff'g because of an equally divided court, In re* Bradley, 600 F.2d 807 (1979).

[39]Schmidt, *Legal Proprietary Interests in Computer Programs: The American Experience,* 21 Jurimetrics J. 345, 358–59 (1981).

[40]Sprowl, A Review of Niblett's *Legal Protection of Computer Programs and* Diamond v. Diehr and *Some Thoughts on Patenting Computer Programs,* 1981 A.B.A. Rsch. J. 559.

[41]*In re* Abele, 684 F.2d 902 (CCPA, Aug. 5, 1982).

was that of a method of calculating the value of certain data and displaying it in a pictorial form. A companion claim which was allowed was the same method, but in which the data was gathered by and displayed on the video display of a CAT scanner. Consistent with *Benson,* this decision restricts patents to specific processes, not overly general concepts. Consistent with *Diehr,* this decision restricts software-implemented processes to situations involving more than calculations—in this case, the image-enhancement techniques used to display output which is tied to the CAT scanning environment.

In another case, the CCPA allowed process claims for compiling computer programs and automatically rearranging various formulas in the programs for calculation.[42] While the invention in this method was to manipulate mathematical algorithms, it was held not to be a nonstatutory algorithm claim because it related to the internal operations of a particular computer and governed the manner in which the computer executed its programs. Again, this decision reflects the new approach—a specific software-implemented process affecting a particular physical environment, and not merely calculating.

In a third recent case, the CCPA allowed claims which heavily involved mathematical calculations in the context of seismic data processing.[43] Although the claims directly recited mathematical algorithms, the claimed process took sonic, seismic data and converted it into another form. This is signal processing tied to and affecting material objects (analog data) in a specific environment.

The Patent Office, in light of *Diehr,* has decided that it now has been given judicial approval to issue program patent examining guidelines.[44] These guidelines endorse the Freeman-Walter two-step test. Instead of concentrating on 35 U.S.C. Section 101 (nonstatutory subject matter) for rejecting software claims, the Patent Office will concentrate on Section 102 (novelty), Section 103 (obviousness), and Section 112 (proper disclosure).

2. Analysis

a. *Mathematical Algorithms?*

The CCPA has had an undue fixation upon mathematical algorithms. This may reflect the archaic view that programs are "scientific" and "mathematical" in nature, or somehow intangible or "mental." It may also reflect the blind leading the blind: the CCPA attempting to generalize from Supreme Court decisions, notably *Benson,* in which the Supreme Court clearly stated it did not feel itself competent to establish policies in this area.

A careful consideration of the CCPA's decisions, however, indicates a different and nontheoretical reason for the CCPA's fixation on mathematical algorithms. The CCPA has been striving to establish clear and broad grounds for

[42]*In re* Pardo, 684 F.2d 912 (CCPA, Aug. 5, 1982).
[43]*In re* Taner, 681 F.2d 787 (CCPA, Jun. 10, 1982).
[44]MANUAL OF PATENT EXAMINING PROCEDURES § 2110.

patenting software. By focusing upon mathematical algorithms, the CCPA has narrowly defined the grounds for rejecting software claims as nonstatutory subject matter.

Still, the CCPA may have picked the wrong narrow definition. It is not the nature of the mathematical algorithm itself which makes it undeserving of protection, but the effect of granting the patent, that is, whether the protection provided by a patent would be too broad. An algorithm is not so much a "mental process" or a "law of nature" as the realization of a solution to a mathematical problem. An algorithm may be compared to processes which can be patented. Some algorithms, like the one in *Benson,* are so primitive or so broad as not to deserve protection. This is not new law; in *O'Reilly v. Morse,*[45] the Supreme Court rejected a claim of Samuel Morse which went beyond a specific invention, the telegraph, and attempted to capture a phenomenon of nature, the transmission of information through electromagnetism.

In addition, the CCPA may have erred in focusing on *mathematical* algorithms. Computers can also manipulate text (word processing), files (data base processing), and graphics (videogames). All such manipulations involve symbolic logic (whether of numbers or other symbols). The principles underlying *Morse* and *Benson* as to patenting of laws of nature apply with equal force to all symbolic logic, not merely mathematical logic.

The further aspect of *Benson* regarding preemption also applies to all symbolic logic underlying internal routines of software. The constraints of computers and computer languages may allow only one of several ways of implementing the same symbolic logic. For example, an operating system which allows a computer to operate with many different users at the same time requires a common and basic internal structure. Each user will put different demands on the system. If the computer were to wait to finish each user's work before going to the next user, there would be interminable delays. Instead, the multi-user operating system "time slices" such that it operates on one user for a certain number of instructions, and then it moves to the next user, and so on, eventually coming back to the first user again. Properly done, the delays that any user notices are negligible. For policy reasons, it might be troublesome if the first manufacturer which invented such a multi-user system could patent it and preclude all others from using that concept without payment of a royalty—but that is the whole point of patent protection. The doctrines set forth in *Morse* and *Benson* serve to limit this problem, but again, this is not new law designed especially for software claims.

b. *The Doctrine of Equivalents*

The "doctrine of equivalents" is that a patent covers equivalent configurations, even though they are not disclosed or actually claimed, unless a purportedly equivalent configuration performs the prescribed function in a substan-

[45]56 U.S. (15 How.) 62 (1853).

tially different way.[46] Thus, if a program could theoretically be reconfigured into a hard-wired machine, then under this doctrine a patent of the machine would protect the program.[47]

This doctrine suggests that the Freeman-Walter test and its undue fixation on mathematical algorithms contains the following loophole: to avoid the Freeman-Walter test, an astute patent lawyer could describe the patent in terms of an apparatus (a hard-wired version of the software) and later use the doctrine of equivalents to prevent software versions of the apparatus from being manufactured and sold.[48] By limiting consideration of the fundamental policy question regarding the scope of protection to situations involving mathematical algorithmic software (as opposed to hard-wired machines), the Freeman-Walter test may allow patents to be issued on machines which essentially have the same undesired ability of capturing or preempting the use of a mathematical algorithm or fundamental process that has been precluded in mathematical software patent claims.[49]

A more fruitful approach for the CCPA would be to determine whether the program being claimed would be patentable when reconfigured into an integrated hardware/software machine (alone in apparatus claims or as part of a process in method claims).[50] Programs which simply calculate primitive algorithms are unworthy of patentability since they merely create electronic calculators. Programs which do more, such as in *Diehr* or in the recent CAT scanner case, could be patented. This approach would also solve another problem which has plagued the Patent Office, that of not having much prior art from which to evaluate software claims. There may be substantial prior art in other computers and in old electromechanical devices which the new programmed computers replace.[51]

c. *The Patentability of Read-Only-Memory*

The confusion over the nature of software and suitable patent analysis can lead to superficial distinctions. A series of distinctions have been proposed over the last decade; most have been disposed of. One remaining distinction

[46]4 D. CHISUM, PATENTS § 18.04 (Bender 1982).

[47]Conversely, if the hard-wired machine patent overly protected a fundamental mathematical algorithm, it would be disallowed. *See* Arshal v. U.S., 621 F.2d 421, 427 (Ct. Cl. 1979), *cert. denied*, 449 U.S. 1078, *rehearing denied*, 450 U.S. 1050 (1981) (analog computer).

[48]*See, e.g., In re* Noll, 545 F.2d 141, 147, 149-50 (CCPA 1976).

[49]See, *e.g.*, *In re* Bradley, 600 F.2d 807 (CCPA 1979), *aff'd, sub nom.* Diamond v. Bradley, 450 U.S. 381 (1981) (rejects Board of Patent Appeals attempt to reformulate apparatus claims as "mathematical" on grounds all computers fundamentally operate based upon mathematical principles); *but see* Arshal v. U.S., 621 F.2d 421, 427 (Ct. Cl. 1979), *cert. denied*, 449 U.S. 1078, *reh. denied*, 450 U.S. 1050 (1981).

[50]The CCPA has recognized this problem. In *In re* Bradley, 600 F.2d 807, 812 (CCPA 1979), *aff'd sub nom.* Diamond v. Bradley, 450 U.S. 381 (1981), the CCPA distinguished *how* a program operated from *what* it did when considered as an integral unit with the computer.

[51]*See* Digitronics Corp. v. N.Y. Racing Assoc., 553 F.2d 740 (2d Cir.), *cert. denied*, 434 U.S. 860 (1977).

which should be discarded is that software embodied on memory chips called Read-Only-Memory[52] is patentable even though the same software embodied in a different form might be unpatentable. This ROM distinction, understood in the context of earlier distinctions, is wholly superficial.

One argument in the early cases was that since no perceptible change occurs in a digital computer when it is programmed differently, no physical change was actually occurring and software was purely intangible, that is, ''mental.'' But certainly the computer hardware can tell the difference, as it executes different ''mental'' programs differently. The CCPA agreed, noting that a computer programmed in one way is a physically different machine than a computer unprogrammed or programmed differently because memory and other elements are differently arranged; the changes may not be visible, but they do exist and are of course measurable by the machine.[53]

A related argument was the early distinction between apparatus and process claims. Again, perhaps because of the emphasis of the Patent Office on the mental steps doctrine, software, at least in process claims, was considered intangible and ''mental.'' A cogent dissent in *In re Chatfield*[54] should have put this argument to rest, but *Benson* involved process claims and thereafter some software claims more appropriately described as process claims were probably reconfigured into apparatus format.[55]

Even in the recent decision of *In re Walter,*[56] an argument arose relating to the perceptibility of magnetically recorded information. One element in the process in that case was that the result of calculations from a mathematical algorithm were placed upon magnetic tape for further use. This was held not to constitute a sufficient change of an article to a different state to make the use of the algorithm merely one step in a larger patentable process or apparatus. This is an unconvincing distinction. Under the doctrine-of-equivalents analysis proposed above, a better distinction is to view the software-implemented process as a ''black box''—a process implemented in some undetermined fashion on an integrated hardware/software machine. In *Walter,* regardless of how the machine stored or displayed its results, it did no more than calculate. Unlike other cases involving similar data in which the raw data itself was processed, in *Walter* the results of such processing were merely further interpreted mathematically.[57] Thus, the process in *Walter* is unpatentable.

[52]ROMs have several progeny, including PROMs (Programmable ROMs), EROMs (Erasable ROMs), EPROMs (Erasable PROMs) and EEPROMs (Electrical EPROMs). ROMs are distinguished from volatile or dynamic memory chips, often called RAMs, meaning Random Access Memory, in that ROMs retain their content when power is turned off, and ROMs are not erasable except, in the case of an EROM or EPROM, by a process separate from the normal memory addressing operations of a computer. With advancing technology, ROMs may be less distinguishable from RAMs. Several companies are now developing NOVRAMs, nonvolatile RAMs.

[53]*In re* Bernhardt, 417 F.2d 1395 (CCPA 1969).

[54]545 F.2d 152 (CCPA 1976), *cert. denied,* 434 U.S. 875 (1977).

[55]Possible examples include *In re* Noll, 545 F.2d 141 (CCPA 1976) and *In re* Bradley, 600 F.2d 807 (CCPA 1979), *aff'd sub nom.* Diamond v. Bradley, 450 U.S. 381 (1981).

[56]618 F.2d 758 (CCPA 1980).

[57]*See In re* Taner, 681 F.2d 787, 790 (CCPA, Jun. 10, 1982) (distinguishing *Walter*).

Finally, the most recent proposed distinction is that of ROMs. The case of *Diamond v. Bradley*[58] can be interpreted to establish that ROM-implemented processes are patentable subject matter even if the same process, software implemented, would not be. This would be silly. The ROM in *Bradley* could have been replaced in the computer with a volatile RAM (read-write) memory chip which looks like the ROM; the software could be first read into the RAM, then the process performed. Protecting a process based on such a trivial technological distinction is unwarranted. A more valid distinction is that the software-implemented process in *Bradley* did more than calculate.

d. *The Arguments Against Patents*

There are many arguments against relying upon patent protection of software. Despite the attention given to the patenting of software, it is unimportant for the vast majority of software vendors. Only a minute number of programs (perhaps less than 1 percent) are inventive enough to be patented. Most software is mundane, automating the same types of accounting or other functions once done manually. Many programs have short product lifespans due to rapid technological advances. Patenting is simply not useful for their protection.

The process of patenting is also unattractive. It is expensive and slow. Congress and the Patent Office have been quite concerned about the patent system's administrative viability.[59] Patenting is also unreliable. Perhaps as many as one-half of all patents which are challenged will be found unpatentable.[60] Even after appeal, the results can be unsatisfactory; the federal appellate courts have varied greatly as to how often they validate patents.

The Patent Office may have been correct in opposing the patenting of programs. The prior art is very recent and largely unavailable. Most software is protected by trade secret protection and, therefore, there are restrictions on its disclosure and promulgation. Also, full disclosure of patented programs may be dangerous. Misappropriation is difficult to discover and disclosure could give free rein to competitors.

The symbolic nature of software which allows the rapid creation of software may have fundamentally changed a premise of patent protection: that the slowness and expense of technological advance requires a broad, monopolistic protection. The very success of the software industry without patent protection seriously calls into question the need for such protection for software itself or for its internal routines. A new program for an existing computer can now be created in weeks or months, and changed as rapidly, and any novelty internally in the program is most vulnerable to copying and not reverse engineering.

In addition, because of the symbolic nature of software, a patent of software may inherently have broader scope than would a patent of a hard-wired

[58]Diamond v. Bradley, 450 U.S. 381 (1981). *See* Schmidt, *Legal Proprietary Interests in Computer Programs: The American Experience*, 21 JURIMETRICS J. 345, 359–361 (1981).

[59]1981 CLA Transcript at 169–175.

[60]*See* Stern, *ROMs in Search of a Remedy: Can They Find It?*, 1 COMPU., L. REP. 4, 8 (1982) (that 70 percent of all patents at the appellate level are held invalid).

machine. A machine can be torn apart and rebuilt; infringement can be appreciated because form and function relate directly. A process which affects materials and produces physical results can also be similarly understood. Because it is symbolic, software is not as easily understood and appreciated. A different program may innocently contain the same internal algorithms and yet perform different functions. In planning and advising clients, lawyers may have a difficult task in determining the scope of a patent on an internal software routine—and a conservative reaction would be to avoid infringement. Thus, such software patents may be given broader interpretation than is appropriate.

More importantly, the number of persons unexpectedly affected by such a software patent may be much larger than is traditional. Software and the computers on which it is operated are separate items. Computer purchasers buy "blank" programmable machines without considering all of their particular uses. Because of a routine inside a program innocently licensed or developed, computer users may find themselves infringing a patent of which they had no knowledge or concern, or may find their options to acquire certain software applications limited.

e. *The Growing Importance of Patents*

The arguments against patenting software can seem compelling, even overwhelming. Nevertheless, patent protection of software is of growing importance. Techniques for drafting acceptable software claims are becoming more widely known.[61] The experience being gained by patent lawyers should allow them to give proper advice. The Patent Office has announced publicly that during the Reagan administration it will concentrate on improving its examination and patent application processing for computer-related and software-related patent claims,[62] and such patents may be issued faster. The Patent Office examiners are gaining more software-related prior art against which to evaluate claims. Most importantly, the federal courts can be expected to become much more receptive to upholding software claims; all patent appeals from district courts will be presented to the CCPA (now called the Circuit Court of Appeals for the Federal Circuit).

In addition, the arguments against patenting software which are based upon its symbolic nature do not encompass the types of software suitable for patenting. Instead, they apply to software which is obvious or not novel enough to be patentable—software which is relatively easy to create or which does not implement transactions in a fundamentally new way.

The fight against patenting software, waged since the mid-1960s, has apparently been lost because of *Diehr.* In addition, advocates of patenting software today have a favorable political and economic climate. The major inde-

[61]*E.g.*, *In re* Sherwood, 613 F.2d 809, 816–17 (CCPA 1980), *cert. denied,* 450 U.S. 994 (1981) (seismic analysis claims drafted as apparatus not process claims with minimal emphasis on the extensive use of mathematical calculations).

[62]1981 CLA Transcript at 169.

pendent users, such as oil companies, have always favored patents on software. Now the major manufacturers also see the advantages of patent protection. In the last decade the price of hardware has dropped considerably. As a consequence, much of the profit in the computer industry now lies in software. Also, many manufacturers of computer hardware benefit greatly by the ''lock-in'' effect which occurs when their customers have substantial investment in software which can only be used on one manufacturer's machines. If essential types of software, such as operating systems and compilers, were available to execute the same applications programs on some other manufacturer's equipment, the user would not be ''locked'' into purchasing the first manufacturer's equipment. Conversely, by taking steps to protect its software and preserve the lock-in, the manufacturer solidifies its customer base. Therefore, the manufacturers, as well as the independent software users and vendors, now favor protecting software.

3. Conclusion

The lesson of *Flook* and *Diehr* is that software must do more than calculate to be patentable subject matter. A program which assimilates data and calculates, even if it is used in a larger process, but which does not automatically implement the calculation in the process (requiring instead a person to interpret the calculation), is not patentable under *Flook.* What is being claimed would be no more than the underlying algorithms limited to a specific field of use. To the extent the software is mechanically or automatically connected to the surrounding physical environment, it is patentable subject matter.

This distinction—calculation versus interconnection—is the basis for determining patentable subject matter regarding software. As framed, however, the distinction still is inadequate. *Calculation* connotes mathematics; as noted above, it should be understood to include all aspects of symbolic logic implemented by software. *Interconnection* connotes use of the software in larger processes, but not software *per se;* as noted above, applying the ''doctrine of equivalents'' means software operating on a computer when considered as an integrated hardware/software unit might be patentable as an apparatus. Conversely, merely tying software to an environment as part of a larger process may not be enough; the software although interconnected may still do no more than static data assimilation or calculation.

The key distinction is the *dynamic* use of the software; its ''real-time'' use tied to and *affecting* the environment. The environment can be affected by the program's output if the output manipulates the surrounding machinery. In *Diehr,* this occurred by a servo-mechanism. In *In re Chatfield,*[63] this occurred internally in the computer. The patented software-implemented process measured for bottlenecks in a computer's operation and then automatically adjusted

[63]545 F.2d 152 (CCPA 1976), *cert. denied,* 434 U.S. 875 (1977).

priorities of users to reduce the bottlenecks. Because of the latter element, the program was not merely gathering and reporting information but was acting upon the information.

The environment can also be affected by calculation alone, if the data is one-time or analog in nature. The analog data, not surrounding machinery, is the affected environment. In the seismic processing cases,[64] for example, programs accomplishing complex mathematical transformations were used to separate the signal from the noise on sonic, seismic data tapes and to produce cross-sectional maps of subterranean features. Because of the feature of signal processing tied to the seismic environment, the claims fell within statutory subject matter. The software affected the environment because it enhanced analog data—data captured on tape from real-time recording devices, not static data placed on tape from manually derived printouts.

This ''real-time'' test and these cases nevertheless still do not establish the patentability of process claims based upon software *per se* or based upon internal routines, algorithms and programming techniques in software. Such claims do not dynamically affect the environment. Instead, the external attributes of software can be patentable subject matter—the manner in which the software is used in an environment. Process claims involving software would be allowed as long as the program in the process automatically and mechanically creates specific and definite changes to analog data or to other machinery or objects. It is this external novelty—new uses of software, mechanically or electronically tied to and affecting an environment through dynamic, real-time processing—which should be protectable by patents. The primary protection of software *per se* and its internal routines would be copyright and trade secret laws.

COPYRIGHT

Software seems like a good candidate for copyright protection. Although it is used in machinery (computers), it is written and not built. Copyright protection seems like a better protective scheme than patent protection for such writings. Under the 1976 Copyright Act,[65] copyright protection attaches automatically but does not have the potentially overbroad scope of protection feared by the Supreme Court in *Benson.* There are serious obstacles to copyright protection, however.

1. The Subject Matter Debate

The Copyright Office has been registering programs since the mid-1960s, but has expressed doubt as to whether software, at least software in certain forms, is copyrightable subject matter.[66] There has been little dispute that

[64]*See* cases cited at note 22.

[65]17 U.S.C. §§ 101 *et seq.*

[66]Office of the Register of Copyrights, Announcement SML-47 (May 1964); Copyright Office Circular 31D (Jan. 1965):

source code is copyrightable subject matter. The recurring question is whether the binary form of software (particularly when it embodies object code) is appropriate for copyright protection.

Binary form is unintelligible to human beings unless it is printed out in some fashion. Under the 1909 Copyright Act, which incorporated the decision in *White-Smith Publishing Co. v. Apollo Co.*[67] in which a player piano roll was held not to be a "copy" of sheet music, unperceptible forms of works were not copyrightable. This decision had its most important application in the music industry, in which phonograph records for many years were not protectable *per se* until the 1909 Copyright Act was amended in 1971.[68] This doctrine also applies to binary form, which, like music on phonograph records or cassette tapes, embodies information in an unintelligible format. A person looking at magnetic storage media or into the memory of a computer will not see any change as the program itself is changed in that memory. Under the 1976 Copyright Act, however, works which are perceptible "with the aid of a machine or device" are now copyrightable and this aspect of the *White-Smith* case has been overruled.[69]

There is a second, more fundamental reason why binary form may not be copyrightable subject matter. It is not "literary" in the commonsensical meaning of that term. It is not designed to be perceived by a human, with or without the aid of a machine or device. Its function is to substitute for manual labor. It is designed to be a mechanical apparatus which runs a machine. Indeed, when binary form is embodied onto ROMs, even more than when it is embodied on magnetic media or other volatile forms, it appears to be a machine element. ROMs are indistinguishable from most other elements of hardware.

Unfortunately, much of the debate on these issues has been sidetracked by

---

The registerability of computer programs involves two basic questions: (1) whether the program is such as a "writing of an author" and thus copyrightable, and (2) whether a reproduction of the program in a form actually used to operate or to be "read" by a machine is a "copy" and can be accepted for copyright registration.

Both of these are doubtful questions. However, in accordance with the policy of resolving doubtful issues in favor of registration whenever possible, the Copyright Office will consider registration for a computer program as a "book" in Class A if:

(1) The elements of assembling, selecting, arranging, editing, and literary expression that went into the compilation of the program are sufficient to constitute original authorship.

(2) The program has been published, with the required copyright notice; that is, "copies" (i.e., reproductions of the program in a form perceptible or capable of being made perceptible to the human eye) bearing the notice have been distributed or made available to the public.

(3) The copies deposited for registration consist of or include reproductions in a language intelligible to a human being. If the only publication was in a form that cannot be perceived visually or read, something more (e.g., a printout of the entire program) would also have to be deposited.

The Copyright Office accepted approximately 2,000 programs for registration under the Copyright Act of 1909 between 1964 and 1977. CONTU FINAL REPORT at 38 (July 31, 1978). Of these, most were registered by just two hardware manufacturers, IBM and Burroughs. *Id.* at 85.

[67]209 U.S. 2 (1908).

[68]Act of Oct. 15, 1971; Pub. L. No. 92140; 90 Stat. 2541, amending Act of Mar. 4, 1909, ch. 320; 35 Stat. 1075.

[69]17 U.S.C. § 102(a).

misleading distinctions. The ROM distinction, for example, is wholly superficial. The machine element distinction has at times been confused with the separate issue of the relation between object and source code. Other distinctions have been proposed based on a ''communication to people'' requirement, although under the 1976 Act ''reproduction'' is all that is required, not ''communication.'' These erroneous distinctions will be explored in an historical context, and proper analysis will be proposed.

a. *CONTU Report*

Prior to enacting the 1976 Copyright Act, Congress created the National Commission on New Technology Uses of Copyrighted Works (CONTU) to analyze the question (among others) of copyrighting software.[70] A blue chip panel studied this problem for several years. CONTU made several recommendations in its *Final Report*[71] which were embodied into the 1980 amendments to the 1976 Act.[72]

In an extensive dissent, Commissioner Hersey felt binary form was not appropriate for copyright because it was a ''machine control element, a mechanical device'' which on constitutional and policy grounds ''ought not to be copyrighted.''[73] He distinguished written instructions in general, which are copyrightable, on the grounds that although binary form contains instructions, these instructions do not tell how to do something, they do it, and what they do is not communicated to humans. He distinguished musical recordings as communicating, while software only ''utters work.''

Commissioner Nimmer approached this problem from a different perspective. While he concurred in CONTU's recommendations, he was troubled because CONTU had not adequately distinguished software from other technological creations for which copying or misappropriation was a problem.[74] These creations would include chip masks, the photolithes from which integrated circuits (chips) are made. These creations could also include recombinant-DNA, which copies itself. Instead, Nimmer suggested that copyright should exist in binary form not because the binary form itself is a ''literary work'' but because (and only if) its output is copyrightable. For example, a videogame would be copyrightable because its output is an ''audiovisual work.'' Data base management or access systems would be copyrightable, because the collection of items manipulated by the data base management system's software is a ''compilation.''

Nimmer's concern indicates that CONTU did not take a *formal* approach (questioning whether software is a ''literary work'' or an ''audiovisual work'' or some other category of copyrightable subject matter), but recommended a

[70]Act of Dec. 31, 1974; Pub. L. No. 93-53, tit. II; 88 Stat. 1973.
[71]CONTU Final Report (July 31, 1978) [hereinafter referred to as CONTU Report].
[72]1980 Computer Software Copyright Act; Act of Dec. 12, 1980; Pub. L. No. 96-517, § 10; 94 Stat. 3028 (codified at 17 U.S.C. §§ 101, 117).
[73]CONTU Report 69-93.
[74]CONTU Report 66-69.

functional approach. In its *Report,* CONTU noted that software is expensive to make but easy to copy, and to steal, and therefore whatever its nature, it deserves protection. In its recommendations, instead of stating that computer software is one of the categories of copyrighted works, CONTU merely included the following definition of *computer program:* ''A *computer program* is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result.'' The interesting point of this definition is that it is functional. It does not define *computer program* with respect to a certain form of the program or a certain type of program. The balance of the CONTU's recommendations concern certain uses of computer programs which are allowable despite the otherwise exclusive rights of the copyright owner. These relate to certain peculiarities of computer software and their uses and once again indicate a functional approach to the problem as opposed to a formal approach.

b. *1980 Amendments*

On December 12, 1980, the 1976 Act was amended in accordance with the recommendations of CONTU with one change.[75] The amendments are as follows:

§ 101. Definitions . . . A *computer program* is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result.

§ 117. Notwithstanding the provisions of Section 106 [describing the copyright owner's exclusive rights] 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 the continued possession of the program should cease to be rightful.

Any exact copies prepared in accordance with the provisions of this Section may be leased, sold, or otherwise transferred, along with the copy from which such copies were prepared, only as part of the lease, sale or transfer of all rights in the program. Adaptations so prepared may be transferred only with the authorization of the copyright owner.

The one change made by Congress from the CONTU recommendations was to substitute ''owner'' in Section 117 for ''rightful possessor.'' Congress gave no official explanation of this change, but apparently was afraid that if one inadvertently left the program lying around, an innocent finder would be a rightful possessor. It has also been reported that Congress was worried that otherwise Section 117 would not allow licensing of software.[76]

In the sense that these amendments reflect approval of the CONTU report, which recommended coverage of binary form, it can be argued that whether or

[75]1980 Computer Software Copyright Act; Act of Dec. 12, 1980; Pub. L. No. 96-517, § 10; 94 Stat. 3028 (codified at 17 U.S.C. §§ 101, 117).

[76]1981 CLA Transcript at 195.

not binary form was covered by the 1976 Act, it certainly has been covered by the 1980 amendments. The amendments themselves are consistent with this argument. The definition of *computer program* covers binary form by implication since the definition does not limit copyright protection for just certain formats or forms of programs; the definition only requires that a certain function be performed. Also, the right granted in Section 117 (1) to execute the program copy in conjunction with a machine, which of necessity covers the binary form of a program, would be meaningless if the copying of binary form for execution in a machine could be considered unprotected.

c. *Case Law*

The cases have wrestled with the problem of binary form. The question of the copyrightability of binary form was first reached in *Data Cash Systems, Inc. v. JS&A Group.*[77] The lower court, relying upon *White-Smith,* the player piano roll case, held that binary form embodied in a ROM was not a "copy" under the 1909 Act both because it was unintelligible to a human being and because it was simply an element in a machine. On appeal, the lower court was affirmed on different grounds: the ROM in question did not contain the requisite statutory notice.

The first two cases to consider the issue under the 1976 Act essentially distinguished the lower court's decision in *Data Cash* as aberrant.[78] The lower court in *Data Cash* had reached its decision even though that issue had not been briefed or argued by the parties.[79]

However, in *Apple Computer, Inc. v. Franklin Computer Corp.,*[80] the issue was once again raised under the 1976 Act. A preliminary injunction was not granted despite the clear copying of internal programs of an Apple computer because the court had substantial doubt whether operating systems software was copyrightable. The programs were all in binary form, either on magnetic media or ROMs. The court's doubts arose in part because the operating system programs in question were designed not to communicate to a human being but merely to operate a machine in substitution of manual labor.

The arguments of the court in *Apple* were not persuasive in a recent case involving operating systems software.[81] In addition, the *Apple* arguments were essentially overruled by the Third Circuit in a decision three days after the *Apple* decision, in a different case. In *Williams Electric Inc. v. Artic International,*

[77]480 F. Supp. 1063 (N.D. Ill. 1979), *aff'd because of lack of notice,* 628 F.2d 1038 (7th Cir. 1980).

[78]Tandy Corp. v. Personal Micro Computers, 524 F. Supp. 171 (N.D. Cal., 1981) (under 1976 Act, object code on ROM is covered, and if have notice on chip, copyright infringed by reverse engineering the chip); GCA Corp. v. Chance, 1982 Copr. L. Dec. ¶ 25,464 (N.D. Cal., July 12, 1982) (object code is covered by 1976 Act, and registration of source code is sufficient to register the object code).

[79]*See* Midway Mfg. Co. v. Artic Int'l Inc., 547 F. Supp. 999, 1012-13 (N.D. Ill. 1982).

[80]545 F. Supp. 812 (E.D. Pa., Jul. 30), *rehearing denied,* No. 82-2107 (E.D. Pa., Sept. 15, 1982), *appeal pending.*

[81]Hubco Data Prod. Corp. v. MAI, No. 81-1295 (D. Ida., Feb. 3, 1983) (slip opinion, at 11).

Inc.,[82] the court found infringement of a videogame on three grounds. It found that the audiovisual work in the "attract" mode was infringed. (The attract mode is the display of a videogame prior to a person commencing actual play.) It also found infringement in the "play" mode. Finally, because of evidence that the underlying ROM was substantially copied, it found copyright infringement of the underlying program. The court held that the argument that binary form of software on a ROM was not intended to communicate to human beings and was therefore not copyrightable subject matter was artificial and contrary to the 1976 Copyright Act.

Williams, however, does not entirely obviate the arguments in *Apple*. The facts underlying *Williams* do not support the breadth of the language in that case. The software in *Williams* was communicative—a videogame—the type of software for which the *Apple* court would express less doubt as to copyrightability. Indeed, a petition for reconsideration in the *Apple* case following the *Williams* decision was denied precisely on the ground that the utility software in the *Apple* case were very different types of works than the communicative videogame software in the *Williams* case.[83]

The arguments in *Apple* may be raised again from a different angle. The Copyright Clause in the U.S. Constitution extends copyright protection to "writings" of an "author."[84] The 1909 Copyright Act used the phrase "writing of an author" to define copyrightable subject matter. The 1976 Act uses "original works of authorship," including such works as "literary works" and "audiovisual" works, to make it clear that it is defining writings broader than some of the case law under the 1909 Act. "Writings," in other words, now covers all forms of artistic expression, including music and film, which are not expressly written. All these forms, however, have in common a fundamental purpose to communicate to human beings. Thus, a constitutional argument remains that binary form is not covered by the Copyright Clause of the U.S. Constitution. Still, the necessary constitutional authority for the 1976 Act or the 1980 amendments could then be found in the Commerce Clause.[85]

d. *Recommended Analysis*

The copyrightability of software is easy to demonstrate. The analysis is straightforward. It is consistent with and supported by the 1976 Act, the 1980 amendments and the attendant legislative history. The problems and distinctions which have been raised are not legal points but *policy* questions which should be distinguished from the legal analysis. The straightforward legal analysis consists of determining software authorship, defining software expression

[82]685 F.2d 870 (3d Cir., Aug. 2, 1982).

[83]No. 82-2107 (E.D. Pa., Sep. 15, 1982), *appeal pending*.

[84]U.S. Const., art. I, § 8, cl. 8.

[85]U.S. Const., art. I, § 8, cl. 3. *But see* Baker v. Selden, 101 U.S. 99, 105 (1880), (copyright protection does not extend to ideas), incorporated as § 102(b) of the 1976 Copyright Act. Arguably, despite the Commerce Clause, this case is still good constitutional interpretation and the Constitution does not allow Congress to overly extend protection to inventions and writings.

and applying the Copyright Act's definition of *copy* to physical forms of software embodying this expression of authorship.

AUTHORSHIP

The authorship in software is embodied in the original written computer instructions, the source code. This authorship consists of the symbolic manipulation of letters, numbers, and other symbols in accordance with the rules and requirements of computer languages. This authorship is analogous to other symbolic engineering authorship and to literary works, and falls within the definition of *literary works* in Section 101 of the 1976 Copyright Act: " 'Literary works' are works . . . expressed in words, numbers, or other verbal or numerical symbols or indicia, regardless of the nature of the material objects . . . in which they are embodied."

Source code often is written in English-like form, like other literary works (although source code instructions must be written in a very precise fashion to be "read" by a computer and do not involve manipulation of human languages). Some computer languages, notably APL, contain only the manipulation of symbols without this English-like overlay. The machine language itself consists of manipulating ones and zeros, again without a further English language overlay.

This authorship, whether in English-like computer languages, in wholly symbolic or arbitrary languages or in machine language, is nevertheless quite suitable for copyright protection. In all cases, the computer languages used are *symbolic.* Originally the ones and zeros of object code represented switches in the computer which actually turned on and off. Today, they are simply symbolic representations of instructions which the computer uses later to turn on and off specific switches and circuits. These symbolic languages are comparable to other symbolic languages, including human languages and (to a lesser extent) engineering "languages" used to construct engineering diagrams and descriptions such as schematics of electronic circuits, blueprints, and other symbolic representations of machines. A computer programmer is not a computer engineer, just as an architect or structural engineer who produces copyrightable engineering drawings and descriptions is not a builder and does not actually weld the steel or lay the foundation.

This is the fundamental principle for distinguishing software from other technology which is expensive to create but easy to copy (the concern raised by Commissioner Nimmer in the *CONTU Report*). So long as the authorship consists of manipulation of symbols and not the actual hard-wiring of a machine, programmers are authors of works comparable to other copyrightable works. The fundamental distinction between building a machine and writing a program is that the program is written in a symbolic language whereas the machine is what it is. To "read" a machine, one would have to "reverse engineer" it in order to understand the creativity which went into building it. To read a program, one need only understand the computer language in which the software was written. The activity of writing a computer program is a different type of activity than building a machine. The activity of reading it is a different type of activity than reverse engineering a machine. The creation of computer programs consists of manipulation of symbols comparable to the creation of other copyrightable works. The reading of computer programs consists of reading of such symbols comparable to the reading of other literary works written in languages other than generally understood human languages.

EXPRESSION

Copyright protects expression, not embodiment. Copyright protects the writing, not the book; similarly, it protects the program, not the ROM, the diskette or the printout. The fundamental principle for the protection of object code and all other program codes derived from source code is based upon the nature of the ''expression'' of the original authorship of the source code. The expression of a book does not consist of the precise language in the book, but consists of the specific interrelation of plot, character, theme, and other elements of the story. Similarly, the expression in the software does not consist of the exact written source code, but the specific logic and design of the program.

In this respect software is more deserving of copyright protection than are other engineering creations such as schematic diagrams, structural engineering descriptions and architects' drawings. There is more freedom of expression of a programmer than in most other engineering disciplines in which engineering diagrams, descriptions, flow charts or other symbolic works are created prior to making the engineering structure involved. In software, there can be many ways to achieve the same processing of transactions. There are often several ways to write individual routines to accomplish specific purposes. There are also individual styles of the manner of coding, such that, like an e.e. cummings poem, even the layout of the source code on printout looks very different depending upon a programmer's style. (Much of the effort in developing better programming techniques is to remove many aspects of individual style from a program in order to make it more universally understandable and useable by other programmers.) Within the constraints of computer languages, which are much more precise than human languages, computer programmers can show a wide range of creativity roughly comparable to that shown by authors of literary works.

This expression is not the functioning of the computer but a symbolic representation of that functioning. The authorship of software should not be confused with its functioning or its use. The expression is the design, logic, structure, and specific choices as to what that functioning will be when the program is run, but not the functioning itself. This is true of all comparable engineering creations. For example, the expression of a schematic diagram of an electronic circuit represents how the circuit will function when built, but it is not the functioning circuit itself.

Much of the confusion in the debate over the copyrightability of software involves the translation of source into object code. The *code* has changed; the ones and zeros of object code are very different than the letters, numbers and symbols of source code. But the *expression* remains the same; only the size of the alphabet has been reduced. Source code is *mechanically* translated into object code through translating programs called ''compilers'' or ''assemblers,'' depending upon the nature of source code. Compilers are designed to carry the same specific logic of source code into object code. Therefore, the *expression* of software is precisely translated from source code into object code.[86]

One reason for the object code/source code problem is that the expression of the program in object code is less intelligible than the expression in source code. This is due to the nature of computer languages. Much of what makes source code intelligible in a higher-level computer language consists of English-like language labels and names, an easy-to-follow structure, an efficient and concentrated set of instructions (so that one instruction in source code may generate a number of instructions in object code), and the ability to include with the computer instructions nonfunctionable comments which can describe what each computer instruction is doing.

Nevertheless, object code is not unintelligible. Indeed, it has no purpose when printed or displayed except to be deciphered. In addition, object code can be almost directly translated into assembly language (even if the original source code were in some other language). Assembly language consists of the object code computer instructions represented by ones and zeros translated into simple mnemonics which are easier to understand. Thus, object code could be rendered more understandable through the aid of machine or device, a computer which translates from object code into an assembly language.

''COPIES''

There is little question that a printout or handwritten coding sheet embodying source code or object code is a ''copy'' of the respective code within the meaning of Section 101 of the 1976 Copyright Act. There should be little question that, based on the language of Section 101, binary form is also a ''copy'' of the source code or object code it embodies. The definition of *copy* in Section 101, and the further description of *authorship* in Section 102(a), state that copyright protection subsists in authorship fixed in material objects from which the work ''can be perceived, reproduced or otherwise communicated, either directly or with the aid of a machine or device.'' In other words, the tangible medium of expression of authorship does not matter so long as the authorship is at least reproducible from it with the aid of a machine or device. Software in-

[86]Source code tends to be written for one particular compiler or assembler. If so, and if it is later used on a different compiler or assembler, there may be minor differences in how the other compiler or assembler translates the source code into object code or assembly language. The expression is still virtually the same. It is possible with many computer languages to reverse object code back into a form of source code. This quasi-source code would not necessarily look the same as the original source code, but would contain virtually the identical specific logic and design which is the expression of the program.

herently can be reproduced from binary form because of the nature of computers. Binary form does not consist of the hard-wiring of a machine; instead, it consists of memory in which the program is stored. It is inherent in any memory device that the information in memory is reproducible, with the aid of a computer, even if it consists of software. This is a fundamental basis for a general-purpose programmable computer.

Because a ROM from the outside looks like any other chip in a computer, it is easy to argue that a ROM should be considered part of the hard-wiring of the computer. Hard-wiring of a computer should be clearly distinguished. A ROM is a memory device. Binary form in a ROM can be placed inside the computer or can be attached externally in a cartridge, as on many home computers or videogame computers. In either situation the binary form can be directly addressed as it would also be addressed if it had been read from a diskette into the internal memory and then used. Binary form on magnetic media, such as diskettes, must first be read into the internal memory of a computer before being used. Software can also be output from diskettes or ROMs just as any other memory can be output. Because of the frequent piracy of software, technological means are often used to restrict the ability to copy programs from diskettes or from ROMs. These technological steps do not make the ROMs or diskettes so "hard-wired" that the software is not reproducible from them, or else the computer itself would be unable to read its own software. In all cases the software is embodied in memory devices from which it can be automatically reproduced. In none of these cases is the software so embedded into the hard-wiring of the computer that it cannot be automatically reproduced.

The printing or display of software from a ROM or diskette onto a different medium of expression, such as onto paper or a video terminal, is sufficient "reproduction" within the definitions of *copy* or *authorship* above. There is no requirement within these definitions that the reproduction be in the same medium of expression. The reproduction from a ROM onto some other medium of expression should be distinguished from the manufacture of an identical copy of the ROM itself. Similarly, the copying of software off a diskette onto a form perceptible by a person is sufficient within the meaning of the Copyright Act without requiring the person to manufacture from scratch a diskette with the same magnetic coding on it.

SUMMARY

This analysis establishes the copyrightability of all types of software (operating systems or applications) in all languages (high level, assembler or machine), in all codes (source or object), and in all forms (written, printed, in ROM or on diskette). The original written source code is the authorship; the program (the logic and design of the software) is the expression; and all forms of software from which a version of the program can be perceived, reproduced or otherwise communicated with the aid of a machine or device are protectable "copies." Unless there are compelling policy limitations, all forms of software could therefore be protectable under the 1976 Copyright Act.

2. Policy Limitations

There are several policies in the copyright laws which may exclude certain types of programs from copyright protection.

a. *Expression of an Idea?*

Section 102(b) of the 1976 Copyright Act expressly denies copyright protection to such things as "ideas" and "processes" in any copyrighted work. This section embodies the prior decision of *Baker v. Seldon*[87] to the same effect. Normally, this doctrine should not limit copyright coverage of software. "Ideas" in the sense of Section 102(b) are different from the intangible "expression" of a program. For example, consider the expression and ideas involved in *Apple v. Franklin, supra.* Franklin did not use the ideas in Apple's operating system programs in order to create its own original works. Instead, Franklin precisely copied the expression as embodied in binary form. This distinction is critical to understand the underlying policies in this area.

Apple was not seeking to protect the *ideas* involved in creating a computer which can run programs that now run only on Apple's (and Franklin's) computers. The "ideas" in Apple's operating systems include such elements as: the manner in which the operating system reads information from or stores it on a diskette and the manner in which the operating system translates programs written in certain computer languages; in other words, the *interfaces* between parts of a computer, not the internal design of the software itself. Apple did not claim copyright of these interfaces. Franklin is not precluded by Apple's copyrights from building a computer based upon Apple's design and upon these interfaces; Franklin could have written its own specific operating systems software to enable its computer to run programs which run on Apple's computers.

It is possible that in some cases the number of ways in which software could express these uncopyrightable ideas would be so limited that any program written to accomplish them would look "substantially similar" to Apple's programs. This is the crux of the idea/expression dichotomy as to software. If the expression captures the idea, it may be unprotectable. In *Morrissey v. Proctor & Gamble Co.*,[88] copying of a set of rules for a sweepstakes contest was not held to be an infringement because there were only a small number of ways to express the ideas embodied in the sweepstakes contest rules. This decision is similar to the patent decision of *Gottschalk v. Benson, supra,* which held that fundamental mathematical algorithms were not patentable subject matter because their patent could effectively preempt use of the algorithm for other purposes.

These arguments may apply with additional force in the area of copyrightability of programs. For example, a copyrighted industrial process program may monopolize the process. Similarly, a copyright on a particular type of op-

[87] 101 U.S. 99 (1880).
[88] 379 F.2d 675 (1st Cir. 1967).

erating system program for the functioning of a computer may monopolize certain key elements in the operation of the computer; in other words, because of the nature of computers, there may only be a limited number of ways to write a program which performs certain utilitarian functions and a copyright for such a program may effectively monopolize the area.

This issue may be reached in *Apple.* In reverse engineering a computer which can operate the same applications programs as can an Apple computer, certain utility programs or routines may of necessity be substantially similar to the same programs in the Apple computer. One program in the Apple II computer, which is common to many computers, causes particular concern: the boot-ROM, called the ''Autostart-ROM'' by Apple. This short program performs the narrow function of starting (booting) the computer by beginning operation of the proper operating system program. Thus, as a narrower grounds for decision in the *Apple* case, it might be found that while in general operating systems programs are copyrightable, the specific programs copied were too closely related to the functionality of the Apple computer to be copyrightable. Even then, while this idea/expression doctrine frees one to reverse engineer expression from ideas, it does not allow one to copy slavishly.[89] Therefore, Franklin may win the war (the principle), but lose the battle.

b. *Useful Article?*

The ''useful article'' doctrine is embodied in the 1976 Copyright Act at Section 113 and in Section 101 within the definition of ''pictorial, graphic and sculptural works.'' The doctrine is that useful articles which portray such works are not ''copies'' except to the extent that they embody separable features. This doctrine is inapplicable to software, which is not a pictorial work. Instead of being drawn as a picture, it is written with letters and numbers or other symbols like any literary work. Even if this doctrine were stretched to apply, the expression in software can be separated from the useful article merely by the process of outputting the program to a video terminal or printer. In addition, the binary form alone is not necessarily the useful article; it is the operation of binary form in the computer which is the useful article, but this is not the ''copy'' which is being protected by the copyright owner.

c. *Distinguish Object Code?*

Debate over the copyrightability of certain forms of programs has usually been based on a distinction between object and source code. This distinction is misleading. A program could be written in object code, which would then also be its own source code. Conversely, as computer systems become more ''user-friendly'' or ''people-oriented,'' many programs will not have a distinct object phase except on a transient basis. The source code performs like object code. In

[89] *See* Continental Cas. Co. v. Beardsley, 253 F.2d 702, 705–06 (2d Cir. 1958) (noting that the copyright is valid even if the expression largely captures ideas, but that liability due to infringement beyond exact copying is difficult to establish).

addition, many programs do not have source code in the traditional sense, but a series of simplified commands. These three circumstances raise serious questions as to the viability of the distinction between object and source code.

Some computer languages are interpreted instead of compiled or assembled. Interpreted source code is executed line by line, with only a transient object phase. Compiled or assembled source code is completely translated into object code before execution and has a permanent object phase. Does the direct translation of source code into machine operation make source code a "machine element" and therefore not copyrightable? Such source code would still contain all the English comments and English-like instructions which makes them seem copyrightable in the first place.

A separate situation is created by a program generator or a high-level data base inquiry system (a *data base* is a computerized compilation of information or data). Program generators can create program code from simplified specifications. The programs that are generated are in a sense never written by programmers, and the simplified specifications which generate the program can be quite different from traditional source code. Similarly, data base inquiry systems can allow access of the data by simple inquiries. Like program generators, the inquiries cause programs to generate search-and-recovery procedures not otherwise written by programmers.

It can be expected that the development of even higher-level languages—particularly in conjunction with the creation of artificial-intelligence machines—will further obscure the distinction between source code and object code and will further remove programming from computer users. Is the program created by another program in any sense a "writing" of an "author"? In part it represents the creativity of the author of the program generator; does this mean that the program generator author is a co-author of the resulting work? If the programs being generated are created by artificially intelligent computers, will such computers suddenly have standing to be "authors" under copyright laws? In light of developing software technology, the source code/object code dichotomy is a doubtful source for an answer to the debate over what software is copyrightable.

d. *Distinguish Operating Systems?*

The *Apple* case may lead to a new test for copyrightability based upon whether the program is communicative or whether it merely operates internally in a computer. Certainly if the program's output creates a separately copyrightable work, such as a videogame, the communication requirement is met. But few programs produce copyrightable output. Instead, to apply this new test meaningfully, "communicative" must be interpreted to cover all applications programs which produce output. The types of programs which would not be copyrightable would be those utility and operating system programs that manage the internal operations of a computer. Instead, those types of programs would have to be of patentable subject matter to be protectable; indeed, a substantial number of the cases which have upheld the patentability of programs involve programs which manipulate the internal operations of a computer.[90]

The communicative/utility or applications/operating systems dichotomy, however, does not result in a satisfactory test for copyright protection. The distinction is unworkable and artificial. Applications programs can have substantial amounts of program code relating to the internal operations of the computer; many parts of operating systems programs could be incorporated as part of the applications program. Further, applications programs can cause extensive internal operations to produce only brief output; conversely, operating systems programs often output extensive status information or directions to computer operators, and can respond to commands from computer users and in that sense be communicative or interactive. Finally, applications programs have been designed which monitor *and* affect the internal operations of computers; the monitoring informs computer operators of inefficiencies, and the computer operators can issue commands back to the monitoring program, which then by itself adjusts the operating systems programs. When does the "communication" end and the "utility" begin?

The fallacy in the "communication" argument is that it presumes that the copyrightable work in question is the *functioning* of the program and not the *writing* of it. This presumption misconstrues the meaning of *authorship* under the 1976 Act. The output of a program could consist of a copyrightable work, such as an audiovisual work, but this output is of a different authorship than the writing of a program and would be separately protectable. A program need not produce any output to be protectable; it is sufficient that the original written program is found to consist of authorship, for that authorship is readable in the same way other literary works are readable, as discussed above.

The communicative/utility distinction also misapplies the meaning of *copy* under the 1976 Act. The Act does not require a copy to be communicative. A copy can be in an unintelligible format if the authorship can be "perceived, reproduced, or otherwise communicated from it either directly or with the aid of a machine or device."[91] Perception or communication are not required; reproduction is sufficient. "Reproduction" should not be read out of the Act.[92] At best, only the original program need be in some sense communicative or authored; but this "communication" in software occurs when it is first written. Software is authored in symbolic computer languages which make software readable and usable by programmers outside of any machine utility or communicative output. Both source and object code have this symbolic nature, and

[90]*In re* Bernhard, 417 F.2d 1395 (CCPA 1969); *In re* Chatfield, 545 F.2d 152 (CCPA 1976), *cert. denied,* 434 U.S. 875 (1977); *In re* Bradley, 600 F.2d 807 (CCPA 1979), *aff'd sub nom.* Diamond v. Bradley, 450 U.S. 381 (1981).

[91]17 U.S.C. §§ 101; 102(a).

[92]*E.g.*, Stern, *ROMs in Search of a Remedy: Will They Find It?*, 1 COMPU. L. REP. 4, 7 (1982) (somewhat disingenuously argues for a "communication" requirement based in part upon analysis of a misquote of the definition of "copy" in § 101 in which the word *reproduced* was omitted).

both can be read and understood by skilled technicians. Their later fixation in any form from which they are perceptible only with the aid of a machine or device does not change this.

e. *A "Machine Element" or Utilitarian Use Limit?*

Rapidly developing technology has created the problem of whether authorship can be protected even though it can be used as a machine element. This is a new problem and existing limitations on copyrights do not apply. Should a new limitation be created because of the "machine element" concept alone? The answer to this question is negative, but it requires a precise understanding of the nature of the machine element argument.

Being a machine element is not sufficient to render the form of embodiment of copyrightable expression unprotectable. Film, phonograph records, and other such works are machine elements in the devices which are used to play them. What is different about software is that its function as a machine element can be independent of any "playing" of the expression. The machine element argument can be misleading since it implies that the work is somehow the functioning of the machine, not the written code itself.

The machine element argument is often confused with the separate question of the relation between source and object code, that is, how much of the authorship in source code is contained in object code. To avoid this separate issue, consider the machine element argument when the program is written in object code, that is, when the source code is the object code. The machine element, binary form (electrical or magnetic pulses), is a direct embodiment of the authorship, the program in object code (written ones and zeros). It is clear that this authorship is reproducible with the aid of a machine or device. When the source code is first translated into object code and then into binary form, an issue other than copyrightability of binary form arises.[93] If the source code work being protected is still protectable when translated into object code, the fact that this object code is transformed into binary form and becomes a machine element does in itself not mean that it or the original work become unprotectable so long as it or the original work are still perceptible with the aid of a machine or device.

The machine element argument also is different than, although related to, the communication/utility distinction. Software when put in binary form to be used to operate a computer has no communicative or literary function. The output of the program may communicate, but it would be a separate copyrightable work from the program, if protectable at all. This does not end the analysis, since the work can be read from binary form and the work then communicates. In addition, binary form can contain textual material. Documents on word processors which clearly are copyrightable are stored in the computer in binary form. Source code or even object code when stored in a computer as textual

[93]This issue is the relation of source to object code and perfection and enforcement of copyrights in both forms of software, and is discussed in Part 3d under "Copyrights."

material are in binary form. In other words, the communication analysis must proceed beyond form to function, to how the binary form is being used.

The basic question in the machine element argument is whether when a copyrighted work is embodied in forms whose sole function is mechanical or utilitarian, do those forms continue to be protectable?

This is a novel question. It arises precisely because of the dual nature of software: it is symbolic writing which can mechanically be translated into utilitarian operation. Other symbolic utilitarian works do not have this direct mechanical correlation. Recipes are utilitarian because they are used to bake cakes (or whatever), but a recipe cannot be put in a machine, causing the cake to be baked. Architects' drawings are utilitarian, but do not build structures by themselves. (With the advent of CAD-CAM, computer-aided manufacture and design, such drawings may soon be used to build products and structures automatically.)

This basic question is related to the "useful article" doctrine, but does not fall within it. The doctrine is based on the problem that a copyright of functional design could preclude others from marketing a similar useful object.[94] For example, if Volkswagen could copyright the design of the "Rabbit" automobile, it could have precluded many of the look-alike automobiles of competitors from being marketed. However, this anticompetitive threat does not exist with binary form. There are many different ways to code programs which perform similar functions. It is the functions which are marketed as useful articles, not the code of the program as embodied in binary form. The functions are *not* protected by copyright. The functions are the analogue to a useful article; the code is analogous to the separable, ornamental features which are copyrightable. Therefore, the basic question of protection of binary form when used to operate computers cannot be answered by existing doctrine on works of utility.

The policies behind copyright protection strongly reject creating a new limitation on copyright of utilitarian uses of binary form. The grant of copyright is intended to reward authors and give them an incentive to create. Software source code is authorship, but is seldom marketed as literature. Instead, practically all the reward to software creators, and the basic incentive to create software, is derived from marketing binary forms for use in a computer. Piracy of binary form by straight copying deprives programmers of reward and incentive.

A further interest served by protecting binary form is to allow technology to develop without artificial distinctions The object code/source code distinction is already insufficient to answer questions relating to interpreted source code, program generators and data base management systems. The applications/operating system distinction is also unworkable due to the problem of determining which parts of the program are "applications" and which are "operating systems." Therefore, freezing copyright protection base upon the current understanding of software technology would be shortsighted.

[94]Mazer v. Stein, 347 U.S. 201 (1954).

Rather than create a new limitation, the courts should allow copyright law to develop with new technology. The dual nature of software makes it an extraordinary development in engineering. One could imagine a more primitive computer industry in which software was painstakingly translated "by hand" into the hard-wiring of computers. Instead, this translation is automatic, and it allows the rapid creation of wholly new machines (programmed computers). Copyright could work well to protect the interests of the creators of software. It should be allowed to do so.

Therefore, instead of the source/object or communicative/utility tests, a distinction should only be made when a form of software—source code, object code, diskettes or ROMs—becomes so distant from a "writing" as to no longer be a "copy." Under Section 101 of the 1976 Act, this occurs when the original authorship is no longer perceptible even with the aid of a machine or device. This occurs when software is so hard-wired (beyond ROMs) into a particular computer that it can be read out only with the intervention of human reverse engineering. In other words, an embodiment of software is no longer a "copy" when it no longer can be copied easily.

This result is consistent with the fundamental, functional purpose of copyright protection. It is easy to steal an expression by copying and this discourages the creation of such expressions. Thus, copyright is an appropriate form of protection of such expression from a functional point of view. This situation is met for all forms of software. Nimmer's concern about distinguishing software from other easily copied technology is answered because software has a symbolic "expressive" nature which is lacking in such items as a recombinant DNA or unprogrammed chips. Therefore, software, in all forms, regardless of their utilitarian or machine element purpose, should be copyrightable.

3. The Enforcement Problem

In general, to show copyright infringement, the copyright owner must demonstrate that the alleged copy is "substantially similar" to the original and that the infringer had "access" to and copied the original (although access is often inferred from similarity).[95] The suitability of copyright protection to software is to a large extent dependent upon how well this test works. Properly applied, it can work very well—a conclusion which may be surprising to the many advocates for trade secret protection or a *sui generis* protective system. Still, before this test will be properly applied, it must be appreciated that the *expression* (the logic and design of the software) is tested for similarity, not the specific source code instructions, and that *access* should be demonstrated by more than an inference from similarity of logic and design.

This test should first be understood in context. Originally, copyright was designed to protect publishers who had to pay royalties to authors from other publishers who could print copies of the same books and have a cost savings by

[95]3 M. NIMMER, COPYRIGHT § 13.01[B] (1981).

not having to pay the royalties.[96] In those cases, the concern was slavish copying, for the copiers were seeking to market exactly the same book with the same words and expression. Similarly, much of the concern in the software industry is against exact copiers. It is an easy matter to copy machine-readable forms of software. Such piracy creates identical copies and thus the issues of "access" and whether a copy is "substantially similar" are easy to answer.

In many circumstances, however, particularly when the form of a literary work is translated to a different form, such as from a book to a film, the question of copying becomes very difficult to answer. One could first look at similar dialogue and proceed to similar plot, theme, names of characters and other identifying material. At some point, however, the similarity becomes only at an abstract or fundamental level; for example, in the idea of a detective story with a certain type of plot twist, as opposed to a specific story by a specific author.

A similar problem arises with software when translated from one form to another. Source code for a particular computer in a particular computer language, for example, can be translated to object code, to source code in a different computer language, to object code designed for a different computer, or to any of the above but disguised to look different. Software translation can be mechanical if a compiler or cross-compiler is available, or can be more difficult than translation of literary works if the translation must be done manually. A computer is very literal minded and the instructions in the manual translation must be very precise in order to work.

Protection can also be more difficult for software than books since there can be many ways to achieve the same functionality. When two programs achieve the same results through different logic and design, it can be difficult to determine whether the protectable expression of the original has been stolen or merely the unprotected ideas in it. Therefore, it becomes much more important independently to demonstrate *access*. Computers operate in similar ways. Computer languages have built-in biases which greatly determine the logic and design of a program. Therefore, programs independently written in the same computer language for the same type of computer may look substantially similar if they perform similar functions. Access should *not* be inferred from similarity.

These problems with applying the test for copyright infringement to software, and the importance of determining access and defining expression, will be explored in specific contexts in which slavish copying is not an issue.

a. *Source Code*

Source code normally contains much more than computer instructions. First, source code can contain comments and annotations which are not executed by the computer but which describe the functioning of certain parts of the

[96]Kaplan, *An Unhurried View of Copyright* (1967); Denicoli, *Copyright and Free Speech: Constitutional Limitations on the Protection of Expression*, 67 CALIF. L. REV. 283, 284 (1979).

program. Second, it contains names of subroutines, variables and labels which can be wholly arbitrary as far as the computer program's functionality is concerned but which are often highly individualized to make the program easy to understand. Third, it can contain an easy-to-follow internal structure, particularly if "structured programming" techniques are used. Fourth, the source code can contain data tables setting forth the structure of memory and the dimensions of variables. Upon compilation or interpretation of the source code, the comments are ignored, the names are replaced with symbolic representations, the structure may be changed to a chronological order rather than the easy-to-follow logical order and the data tables are implemented and can be difficult to recreate from object code.

It is a trivial matter to change the comments, the names of routines, variables and labels, and the order of subroutines and other parts of the program structure. It is also possible to rearrange the memory structure or select different dimensions for variables and still achieve the same functionality in many circumstances, or even to incorporate the data tables into the functional parts of the program. These changes can be done in a matter of days or weeks and would have little or no impact on the operation of the program. The new program, however, would not look at all similar. These selections are symbolic and do not have intrinsic meaning to the computer as they do to the programmer. These changes present a problem which is unique to programs among literary works. In other literary works, changes in the structure, names, words, and size cannot be done without significantly affecting the literary work itself.

With proper expert testimony, it should be possible to argue successfully that the source code changed as above contains the same expression as the original program. Once the symbolic nature of many aspects of programming is understood, it can be urged that the parts of the expression which have been infringed are those involving the logical structure as seen by a computer and not as seen by the programmer.

Source code can also be changed beyond simply changing symbolic values. Noncritical subroutines and algorithms can be reorganized. There are often several ways of implementing the same mathematical calculation or the same manipulation of textual, graphic or stored material. These changes may improve or hurt the program's speed or its ability to handle peak loads, but often not noticeably, and should not affect its accuracy in producing results. A copyright injunction may still issue, this time based upon key or complicated routines which remain unchanged, but often with a little more effort even those could be changed.

At some point, a court would probably put its foot down and say that copyright only protects *expression* and if substantial changes are made, only underlying *ideas* are left in the infringing copy. As expressed by Learned Hand: "Upon any work . . . a great number of patterns of increasing generality will fit equally well, as more and more the incident is left out . . . there is a point in this series of abstractions where they are no longer protected [by copyright] . . . ."[97]

b. *Program Description*

A similar problem arises with respect to program descriptions. Program descriptions aid the programmer in writing the source code. Program descriptions include flow charts and pseudosource code, instructions which are similar to source code but do not comply with all the detailed requirements necessary to make the source code work on an actual computer. Such program descriptions, although they could not be compiled on a computer into functioning object code, can be easily translated into working source code by an experienced programmer.

Such program descriptions, if they are as complete and detailed as the source code created from them, are "copies" of the original program because they contain the same protected expression. When the descriptions are less complete, or are written in a more abstract fashion than flow charts or pseudosource code, the question arises as to whether expression or ideas are being described. Indeed, in some circumstances the program description would be represented by a logic table or some other entirely abstract structure. Although these abstract structures denote logically the same functionality as the source code, they appear far removed from it, and they cannot be used as easily by a would-be copier to create the source code. At some point such abstract program descriptions—and the different source code created from them—would likely be found not to be a copy of, and not to infringe, the underlying source code. These abstract forms describe the *ideas* in the program more than the expression. The specific design choices of the programmer—the expression of the software—are not included in the abstract program descriptions.

When the descriptions consist primarily of supporting materials like user documentation and functional descriptions of the program, this same problem arises. Again, these supporting materials do not reveal the manner of programming—the expression—but the underlying functionality. In *Synercom Technology, Inc. v. University Computing Co.*,[98] the use of instruction manuals and input formats to create programs with similar functionality and compatibility was held not to be a violation of copyright since an input format structure is an idea without a separately protectable expression. However, the verbatim use of substantial portions of the instruction manuals themselves was a violation of the original owner's copyright in the manuals.

This result is consistent with the copyright analysis recommended above. The expression of software is its internal specific logic and design, not its external input and output formats and interfaces. The formats would be protectable only as independent works, such as videogames. In addition, protection of the

[97]Nichols v. Universal Pictures, Inc., 45 F.2d 119, 121 (2d Cir. 1930) (L. Hand, J.).
[98]462 F. Supp. 1003 (N.D. Tex. 1978).

expression does not protect the underlying programming techniques or the functions themselves. User manuals normally only reveal input and output formats and techniques for using functions—the ideas, not the expression, of the software. Therefore, a person who only uses user manuals to reverse engineer software is not an infringer, since he or she never had *access* to the protected *expression* of the software.

This result also seems correct on policy grounds. It would be anticompetitive, and probably a mistake, to allow a software owner to prevent copying of formats and functions. This would allow Apple Computer, for example, to preclude others from making Apple-compatible computers, even if they reverse engineered the proper interfaces and not merely slavishly copied Apple's programs (as Franklin Computer did). A different result would allow patent-like monopoly protection over more than expression without the novelty required of patents and without the relatively short protection period granted patents.

c. *Object Code and Decompilers*

Different problems arise with protection of object code. It is extremely unproductive to derive new programs directly from object code except by slavish copying. Instead, object code must first be decompiled or disassembled into a cousin of the original source code. This cousin does not contain the written comments in the original source codes, the same names of variables, routines and labels, the same structure nor the original data tables, since all of this information was (most likely) not retained in the object code. Still, this cousin contains virtually the same expression as the original source code, the same logic and design, since if it is recompiled or reassembled it would operate the computer in virtually the same way as was intended by the original source code. This cousin is a ''copy'' of the original source code.

Therefore, decompilation infringes the owner's copyright. This provides strong theoretical protection to the copyright owner who markets object code in binary form only, without program descriptions other than user manuals. Access to the expression of a program is legally impermissible. To gain access, the object code would have to be decompiled. Object code itself is too incomprehensible to provide effective ''access'' to the expression of the program. In addition, access to object code by printing it from binary form would also impermissibly create a ''copy.'' This situation also provides strong theoretical comfort to the would-be infringer who does not print or decompile the object code but only works from input/output formats and user manuals. Such a person would not have ''access'' to the protected expression, and would not be an infringer.

This result, surprisingly, means copyright law may protect software marketed in object code on binary form better than it protects books or film. Works derived from the latter are subject to fine distinctions which can make legal planning difficult; for example, courts are now attempting to draw such fine distinctions in deciding whether *Battlestar Galactica* is a derivative work of

Star Wars.[99] Works derived from object code, however, are subject to a simple test: was the object code decompiled or not. For example, consider the issues in *Apple v. Franklin, supra,* in this light. Franklin slavishly copied, and should be found to have infringed Apple's copyrights. If Franklin had first decompiled the programs and derived programs which accomplished the same functions, it would have had access and would have infringed Apple's rights. If Franklin had left the internal attributes of Apple's programs alone and instead used the external attributes to create compatible programs, it would not have violated Apple's rights. The case could have been decided upon a single factual issue.

While this simple test provides theoretical comfort to software owners and persons making similar software without decompilation, it may not provide them practical comfort. The difficulty of showing direct copying of traditional literary works led to proof of copying by circumstantial evidence—similarity and access.[100] Elevating the importance of proving access by requiring evidence of decompilation may make such proof less circumstantial and more difficult. Still, unlike most literary works, software can be "fingerprinted" by dummy code or unusual ways of programming.[101] If the infringer goes to the trouble of removing fingerprints, other circumstantial evidence can be used to demonstrate decompilation, such as comparing the amount of time taken to create the original program with the time taken to copy it. In some circumstances it may still be difficult to show access by decompilation, and that difficulty may make enforcement of copyrights in software more troublesome. Nevertheless, these difficulties can be justified by the greater clarity between infringing copies (after decompilation) and noninfringing similar programs (only after reverse engineering). In addition, since computer structure dictates program logic and design to a great extent, a contrary result could grant too much protection to copyright owners.

d. *Object Code and Compilers*

A second problem involving enforcement of rights in object code arises in the process of compilation or assembly of object code from source code. This process is accomplished by separate programs, including a compiler or assembler and linking programs (which link separately compiled source code modules into one executable "load module"). These separate programs almost invariably are owned by another and licensed for use in creating object code. Use of these separate programs in translating source code into object code obscures the legal relation between source and object code. Is object code a "derivative work"? If so, does the owner of the compiler gain co-authorship rights to the object code? The answer to both questions is negative, but the reasoning is complicated, and requires an understanding of the process of compilation.

[99]Twentieth Century Fox v. MCA, Inc., 209 U.S.P.Q. 200 (C.D. Cal. 1980), *rev'd,* 696 F.2d 689 (9th Cir., Jan. 11, 1983).

[100]*See* Midway Mfg. Co. v. Artic Int'l Inc., 547 F. Supp. 999, 1012 (N.D. Ill. 1982).

[101]*See* Part 1 under "Other Protective Schemes."

Upon compilation of source code, most of what makes source code expression appear unique—annotations, names, structure, data tables—is obliterated, as noted above. What remains is largely only the instructions in the source code, reorganized and rewritten in a different medium of expression. The source code has been "translated." This superficially implies that object code is a "derivative work" rather than a "copy." The definition of *derivative work* under Section 101 of the Act includes any "translation . . . or other form in which a work may be recast, transformed or adapted."

The concept of derivative work, however, requires derivation due to creativity, not physical act. Only the added creative elements of a derivative work are independently protectable.[102] Object code is created by physical act—the use of a compiler or assembler. To the extent that there are any independent creative elements in the resultant object code, they are authored by the programmers who wrote the compiler or wrote the various other utility programs that enable a computer to compile complicated programming material.

The independent creative elements in compiled object code, however, are not protectable. They consist of ideas, not expression. The way in which any source code statement (or group of statements) is transliterated into object code is an idea. The automatic implementation of each of these ideas by a compiler is a mechanical process with insufficient originality to be copyrightable expression.[103] The object code fails to contain sufficient independent authorship which could be separately protectable as part of a derivative work. Therefore, the compiler's owner can claim no co-authorship rights in the resulting object code.

What then is the legal relation between source and object code? Source code and object code could be considered part of the same work. Source code is written with object code in mind; both works are "written" together. Object code could instead be considered an encryption of source code and in that sense a copy.[104] Object code is a mechanical and precise translation of parts of source code, an exact transcription of one set of words or symbols into another set. The precise set of instructions in the source code is encrypted in object code.[105]

[102]*See* Gracen v. Bradford Exchange, 696 F.2d 300, 302 (7th Cir., Jan. 12, 1983); L. Batlin & Son, Inc. v. Snyder, 536 F.2d 486, 490–91 (2d Cir. 1976).

[103]In Signo-Trading Int'l, Ltd. v. Gordon, 535 F. Supp. 362 (N.D. Cal. 1981), the list of words translated from English to Arabic by use of an electronic device was held not to be copyrightable since they consisted of ideas and their implementation by computer was too mechanical to involve sufficient copyrightable originality. An automatic process can nevertheless cause a derivative work. In Midway Mfg. Co. v. Artic Int'l, Inc., 547 F. Supp. 999 (N.D. Ill. 1982), a set of ROMs constituting a "speed-up kit" which plugged into a videogame and electronically sped it up was held to create a derivative work—the faster videogame.

[104]GCA Corp. v. Chance, 1982 Copr. L. Dec. ¶ 25,464 (N.D. Cal., July 12, 1982). *See* Reiss v. National Quotation Bureau, 276 F. 717 (S.D.N.Y. 1921) (L. Hand, J.) (code book of coined words copyrightable).

[105]The annotations and comments in source code are not copied in object code. Since this part of source code are irrelevant to the design of the program, they should be considered separate literary works from the program.

There is, however, an even deeper relation between source and object code. It is not the precise written source code which is protectable, but the underlying logic and design, the program. This program in object code has the same logic and design as the source code; indeed, source code is tested by compilation and execution of object code. It is therefore inherent in the relation between source and object code that they contain the same expression—and this expression is what is protected by copyright.

The test for infringement—similarity and access—is as an evidentiary matter one step removed from this deeper relation between source and object code. Similarity of literary works such as books is typically shown by similar words, one step removed from the expression, the interrelation of plot, character, and other elements of the story. The underlying expression traditionally only need be explored when derivative works are in question—when the original words are translated or transformed. Object code similarity will be shown in other ways, such as by direct compilation of the original source code and comparison of this object code with the alleged infringing object code. Where the object code is first decompiled, the cousin-source code modified, and then recompiled, more complicated proof will be necessary. This proof may be accomplished in the same manner as demonstrating that a work is a derivative work. Despite this similarity in proof, object code remains simply a copy of its underlying source code.

e. *Current Licensing Practices*

Copyright alone may not support current marketing practices and assumptions regarding software. The 1976 Act in some respects is designed to protect marketing practices of an age prior to that of Xerox, IBM and Apple.

Under Section 109 of the 1976 Act, the owner of a program copy may resell it or otherwise remarket it. This at least means that a user of a program copy could satisfy its desire to use it and then sell it, removing a potential buyer from paying the author. Software vendors typically grant nontransferable licenses.

Under new Section 117(1), enacted by the 1980 amendments, the ''owner'' of a program copy has the right to copy the software for execution in a machine. Arguably, taking the copy provided by the licensor or seller of the program and copying it into several computers contemporaneously would also be allowed; technically, also allowable would be leaving one of these copies in the machine and selling or giving to a friend the copy provided by the vendor. Each new copy internally used falls within Section 117(1) as being utilized solely in conjunction with the use of a machine. Since most program licenses are priced on a per-computer basis, this interpretation of Section 117(1) would cause substantial financial hardship, especially when the software is licensed to a large company which has many computers. This problem will become even more widespread as ''office automation'' techniques are used by companies, which will result in placing a computer at every person's desk. The *CONTU Report* does not discuss this problem except in the context of selling a single copy and keeping the original.[106] Contexts other than this probably were not considered when CONTU's recommendations for amending Section 117 were drafted.

Section 117(1) also allows a program copy owner to make an "adaptation" of a program if it 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." The *CONTU Report* explained that this right was to enable "conversion of a program from one higher-level language to another to facilitate use" or adding "features to the program that were not present at the time of rightful acquisition."[107] This right appears most suitable to source code, but could apply to object code. To adapt object code such as by adding features may require that the object code be decompiled. If so, it may be legally permissible for a program copy owner to gain intelligible access to the program by decompilation. This would enhance the likelihood of the program copy owner reverse engineering the program and marketing its own version of the program in competition with the original program owner. Section 117 arguably does not allow this result, since the adaptation can be legally used "in no other manner" other than to operate the original program; still, it may be used as a loophole to justify access in a case involving infringement after decompilation.

To avoid problems, program distributors often attempt to limit the right of resale, copying and adaptation by characterizing the original sale as a "license." In the personal computer market, this is done by enclosing a warranty card with the program, encouraging or requiring the buyer to sign and return the card, and including on the card a statement that the program is licensed, not sold. Sometimes the diskette containing the software is shrink-wrapped, and acceptance of the license arrangement is found in the act of breaking the wrap. Although theoretically a license may exist in that circumstance, it is certainly questionable whether such a license would be enforced, particularly in view of the protection given consumers in this country.

Assuming a license is created, it may not be effective against Section 117. While the Section 117(1) exception only applies to an "owner" of a program copy, and not to a licensee, it is unclear what "owner" of a "program copy" means. Possibly, a distinction is intended between the software and the media on which it must reside. If so, an owner of the medium would be a "program copy owner" under Section 117. Thus, if a person licensing the program is nevertheless deemed to be an owner of the medium, that person would have this Section 117(1) right. For many program distributors the safest solution to this problem is to rely upon a combination of copyright and trade secret law. The per-computer limitation, for example, even if unenforceable under copyright law, could be enforced because of trade secret restrictions in the license.

[106]CONTU REPORT at 31–33.
[107]CONTU REPORT at 13–14.

4. Software-Related Works

a. *Videogames*

Much of the recent case law in the area of copyright of programs involves videogames. Most of these decisions concern not whether the program itself has been copied but whether the audiovisual display of the program has been appropriated.

The argument that the videogame constitutes a protectable audiovisual display has allowed several videogame distributors to protect their games.[108] The argument has also allowed game distributors to prevent the importation of infringing games.[109] The game distributors do not always win, because the game itself cannot be copyrighted, only the stylized expression, that is, the precise pattern of the characters and pictures in the display of the game.[110]

One interesting problem is whether the player is a "co-author" of the game. This problem is usually raised as a defense to infringement or as a challenge to registration. The game distributors have to be careful regarding registration. Typically, they submit a videotape of the game-display in its "attract" and its "play" modes; they should specify whether they are registering the videotape as an artistic work itself, the game-display, or the underlying program. The "co-author" challenge arises because in the "play" mode, the game can have a very large number of possible sequences. The videotape deposited upon registration normally would contain only a few of those possible sequences. The argument is raised that only those sequences have been registered, or that the person actually playing the game is a co-author of the particular sequence when it is being played.

[108]William Elec., Inc. v. Artic Int'l, Inc., 685 F.2d 870 (3d Cir., Aug. 2, 1982) (order granting preliminary injunction); Atari, Inc. v. North Am. Philips, 672 F.2d 607 (7th Cir. 1982) (prelim. inj. granted against "K. C. Munchkin" as infringer of "Pac-Man"); Stern Elec., Inc. v. Kaufman, 523 F. Supp. 635 (E.D.N.Y. 1981), *aff'd,* 669 F.2d 852 (2d Cir. 1982); Midway Mfg. Co. v. Drikschneider, 543 F. Supp. 466 (D. Neb. 1981) (order granting preliminary injunction); Midway Mfg. Co. v. Artic Int'l, Inc., 547 F. Supp. 999 (N.D. Ill. 1981).

[109]*In re* Coin-Operated Audio-Visual Games and Components Thereof, Investigation No. 337-TA-87, USITC Pub. No. 1160 (June 1981) (Galaxian); *In re* Certain Coin-Operated Audio-Visual Games and Components Thereof, Investigation No. 337-TA-105, USITC Pub. No. 1267 (July 1982) (Rally-X; Pac-Man). The first investigation found infringement due to similar "attract" modes; no decision as to infringement of "play" modes beyond first few moments, due to co-authorship problem. The second investigation found infringement in both modes, relying on *Stern* and *Atari.*

[110]Midway Mfg. Co. v. Baudai-America, Inc. 546 F. Supp. 125 (D.N.J., Jul. 22, 1982) (sufficient factual questions to deny prelim. inj. to "Galaxian" and "Pac-Man" owner against vendor of handheld versions of those games); Atari, Inc. v. Amusement World, Inc., 1982 Copr. L. Dec. ¶ 25, 347 (D.Md. 1981) (in copyrighting "Asteroids," Atari did not prevent others from making asteroid-like game, only games with "Asteroids" characters and features). *See* Atari, Inc. v. North Am. Philips, 672 F.2d 607 (7th Cir. 1982) (videogames); Durham Ind., Inc. v. Tomy Corp., 630 F.2d 905, 914–15 (2nd Cir. 1980) (toys). *Cf.* Morrisey v. Proctor & Gamble Co., 379 F.2d 675 (1st Cir. 1967) (copyright may not be an infringement when there are only a small number of ways to express an idea, such as a set of rules for a sweepstakes contest). Traditionally, copyright does not protect games. Antimonopoly, Inc. v. General Mills Fun Group, 611 F.2d 296, 300 n.1 (9th Cir. 1979).

This co-authorship question is fundamentally specious. All sequences are embodied in the program; the player only selects one as he or she plays. This is roughly analogous to selecting and viewing parts of a movie, parts of a video-disc or certain slides of a slide show. It can also be compared to skimming through a book. These analogies are not perfect, because the interactive nature of the game is fundamentally different from the passive nature of watching a movie or reading a book. Still, what is protectable is not the game itself but the repeated patterns of characters in the game and other elements of the game, and these repeated patterns do not materially change between different sequences.[111]

b. *ROMs, Chips and Chip Masks*

A peculiar enforcement problem arises when software on ROMs is considered a tangible embodiment of a copyrighted work. One method of copying a ROM is by photographing the inside of the chip to determine the successive layers of semiconducting material and how they are laid out, and to reverse engineer how those layers were constructed. A related way is to acquire copies of the actual "masks" used in creating the chip. Chip "masks" are alternatively transparent and opaque representations of a particular layer of semiconducting material on a chip, which, by a process similar to photoengraving, can be used to create each particular layer.[112] If a photograph of the inside of the chip is not a copy of the program, and if the design of the chip itself is not copyrightable because the design is a work of utility (see below), a loophole may be created for copiers of ROMs. However, this method of copying does not disguise the fact that the resulting chip is a copy of the program and infringes the copyright. Thus, even if the intermediary steps themselves do not constitute infringing acts, the subsequent creation of the chip is the necessary infringing act.

The problems of ROMs should be distinguished from those of chip masks or chips (integrated circuits) themselves. Intel, among other semiconductor companies, has attempted to protect its chip masks and the resulting chips through the use of copyright. Intel attempted to copyright nine mylar masks for its 8755 microcomputer under the category of *technical drawings,* one of the types of "pictorial, graphic and sculptural works" as defined in Section 101 of the 1976 Act which are within the types of authorship subject to copyright under Section 102. Intel had successfully registered the masks as technical drawings but was refused registration when it proceeded to take two of its chips and submit them to the Copyright Office as published copies of the masks. Intel

[111]Stern Elec. Inc. v. Kaufman, 669 F.2d 852 (2nd Cir. 1982).

[112]*See Toward the Silicon Foundry,* 248 Sci. Am. 82 (Feb. 1983); Angell, *Silicon Micromechanical Devices,* 248 Sci. Am. 44 (Apr. 1983); Brady, *The Bumpy Road to Submicron Lithography,* High Tech 26 (Mar. 1983).

brought a mandamus action, but after a few depositions the case was settled and none of these issues were resolved.[113]

Intel was not likely to succeed. Its argument is that the "technical drawings" authorship in the chip masks has been embodied in the chips. Under Section 101 of the Act, however, works of utility created from "technical drawings" or other "pictorial, graphic and sculptural works" are not copyrightable. Indeed, Intel's situation is not even a difficult question. Function shapes the form of the chip; there are no aesthetic or artistic "ornaments." The harder cases are when the work of utility is to be seen by people and is designed to be attractive as well as functional.

Other works of authorship as defined in the 1976 Act do not have this utilitarian use restriction explicitly. Therefore, arguments have been made to bring chips and chip masks under some other type of authorship. It has been half-seriously suggested, for example, that since chip masks are "displayed" in a factory, which can be construed to be a public place, therefore creating a public performance, they are "audiovisual" works and are protected from being "performed" elsewhere in the process of making the actual chips from the copied masks.[114] However, unlike slides, videotapes or film, the masks are not *displayable* through use of the chips. Those parts of the masks which are audiovisual works are not embodied on the chips. Thus, protecting the masks as audiovisual works would not protect the chips.

A more serious possibility concerns the process of creating and using the masks. This process usually includes digitalization of the masks to be stored in a computer for easy re-creation, storage and reference. The data can be construed to be a "compilation," protectable under Section 103 of the 1976 Act, and arguably the actual masks as well as the chips created from the compilation of information are tangible embodiments of it and consist of infringing copies. The problem is that this information may not be recoverable from chips even with the aid of a machine or device without human reverse engineering and therefore the authorship is not embodied on the chips. Again, the compilation may be protectable, but not the resulting chips.

Another interesting suggestion is to copyright the topology of the chip itself as a "sculptural" work. Since the chip's topology is utilitarian, however, arguably only nonfunctional, ornamental parts of the topology (i.e., virtually none of it) are protectable.[115] Still, the topology can be photographed. Blowups of these photographs are multicolored and look "artistic." Would it help the chip manufacturers to hang these blowups in their front lobbies as art? However, then the ornamental or artistic elements would not be in the chips but in the photographs. It would be an artificial result to find an infringement of a

[113]1981 CLA Transcript at 160-61. The case was *Intel Corp. v. Ringer*, No. C-77-2848 (RHS) (N.D. Cal., filed Dec. 16, 1977 and April 13, 1978), *dismissed by stip.* (Oct. 10, 1978).

[114]1981 CLA Transcript at 161-62.

[115]Esquire, Inc. v. Ringer, 591 F.2d 796 (D.C. Cir. 1978), *cert. denied*, 440 U.S. 908 (1979) (austere, nonornamental lamppost not copyrightable).

picture or sculpture in a copy which is too small to see and is sealed from viewing.

Yet another suggestion is to include actual programs or data on the chips. Many chips, specifically microprocessors and microcomputers, contain certain fixed software programs to aid in their operation.[116] Any copier of these chips will be making a copy of the program parts of the chip and therefore will be violating the copyright in the programs. If a chip being designed does not have a built-in program, a ''dummy'' program could be added in the original form of the chip, which would be erased or ignored when the chip was put to functional use. Alternatively, data constituting a ''compilation'' could be entered in the chip, which again would be erased or ignored upon functional use. Unfortunately, even if sound on a theoretical basis, these latter ideas suffer impracticality. The chip infringer could copy only the erased form of the chip, or if the dummy programs or data are not erased after use is commenced, the chip-infringer could copy around the dummy parts. In either case, there is no protection.

The copyrightability of software on chips should be firmly distinguished. Software is a separate, symbolic work. What is the authorship which is fixed on a nonsoftware chip? The engineering of chips themselves does not involve the creation of symbolic material which is later fixed on the chip. While the earlier steps in creating a chip (logic diagrams, component schematics, chip designs, digitalized data of the designs and the masks) may be to some extent authored, this authorship is not fixed onto the chip. These earlier, arguably symbolic works cannot be mechanically ''perceived, reproduced or otherwise communicated'' from the chip. They are either entirely lost or too connected to be separated. With ROMs, however, it is possible to read out the software which is fixed on the chip.

Chips may not need copyright protection. The semiconductor industry competes largely by offering new products and retaining a cost advantage. Costs tend to drop 28 percent for each doubling in volume of production.[117] There may be sufficient incentive to create new chips by being first. Also, the market acceptance of a chip tends to increase when it can be acquired from several sources, and second-source copiers may actually increase the originator's volume and return on investment. From a broader perspective, chip consumers benefit from lack of protection. The threat (or reality) of copiers forces originators to decrease prices to match decreased costs in order to maintain market share. Protection, therefore, may not increase chip variety but would decrease chip sources and increase chip prices. Finally, while chips are expensive to design and less expensive to reverse engineer, ''copying'' is nevertheless an expensive and time consuming process. It is not mechanical, as with software. While chip copying technology may improve dramatically, currently

[116]Patterson, *Microprogramming*, 248 Sci. Am. 50 (Mar. 1983).
[117]Noyce, *Microelectronics*, 242 Sci. Am. 63, 67 (Sept. 1977).

the functional reason for copyright protection—a work is expensive to create but easy to copy—is not strong.[118]

c. *Firmware (Microcode)*

The copyrightability of firmware may test the limits of the "symbolism" analysis proposed above.[119] "Firmware" is microcode, the interface between software written in machine language and hardware operation.[120]

Instructions in machine language are not directly translated into machine operation; the ones and zeros of object code do not represent actual on and off signals to computer circuits. Instead, object code instructions are first decoded and then translated into a different set of binary signals which turn on and off circuits. This different set consists of microcode instructions. One object code instruction may be decoded into a number of microcode instructions which together execute a microprogram. The binary ones and zeros of microcode can be directly connected to control wires; in such cases, the microinstruction 010011, for example, would turn off the first control wire, turn on the second, turn off the next two and turn on the last two. In other cases, the microinstruction is itself decoded to turn on or off specific control wires. This can be done because often several control wires are only used together, or never used together; a shorter microinstruction length than the number of control wires can still contain all possible useful control-wire combinations.

Programming was a major advance in computing. Instead of rewiring machines, only volatile memory location need be changed to change machine applications. Microprogramming represents an equivalent advance in computing. Instead of rewiring the control wires of a machine and changing the machine language, only the microprogram memory need be changed to change a machine's control system. This enabled IBM with its 360 family of computers to offer seven different computers, varying in speed by a factor of 300 and in cost by a factor of 100, which all operated from the same object code instruction set.[121] IBM has been able to continue this compatibility with many newer families of ever-changing computers, including the 370, 4300, 303X and 308X series.

The next trend is to skip the object code "macro" instruction set and design machines to be programmed in the microinstruction set. The programmer for the most part will still write in higher-level languages, which would be compiled into microcode instead of object code. The microcode computers may contain several microinstruction sets, depending upon application. Instead of

[118]The problems of chip makers have been presented to Congress in connection with H.R. 1007, introduced and not acted upon in 1979; S. 3117 and H.R. 7207, introduced and not acted upon in 1981; and H.R. 1028, introduced in 1983 and pending consideration.

[119]*See* Part 1 to "Copyright."

[120]Patterson, *Microprogramming*, 248 Sci. Am. 50 (Mar. 1983). *Firmware*, like *software*, has various other meanings. It can refer to proprietary programs sold only by one computer company (the firm's software). It can also refer to ROMs, being a firmer form of software.

[121]*Id.*, at 56.

being general-purpose computers, they would be specific-purpose computers with several specific purposes. The contents of microcode memory will be changeable to enable each special application to be accomplished with optimal hardware efficiency.

Is a microprogram copyrightable? When written, it can consist of symbols—ones and zeros. It thus could fall within the definition of a "literary work" under Section 101 of the Copyright Act. The critical question is whether it is authored or built—is the microprogrammer manipulating symbols or setting circuits on or off? The answer depends not upon microcode itself but upon how a specific microprogram is authored. If it is done by setting circuits, then the result is not copyrightable. If it is done by manipulations in accordance with a microprogram "language," then it is copyrightable. This means the original set of microinstructions placed in the microcode memory are not copyrightable, but any subsequent microprogram which uses the microinstruction set to perform computer functions is copyrightable. The original creation of a microinstruction set is based upon actual (not symbolic) understanding of machine circuit operation. The subsequent writing of microprograms based upon the microinstruction set is symbolic, since it is a process one step removed from machinery.

5. Publication and Notice

Under the 1909 Act, publication was the point at which copyright attached.[122] Under the 1976 Act, copyright attaches when a copyrighted work is fixed in a tangible medium of expression for more than a brief moment.[123] The concept of publication is still important with respect to the use of a copyright notice. The notice is not needed unless the work is published. However, it is prudent to place the notice even on an unpublished work, although this creates other problems, as will be discussed below.

a. *Limited Publication*

It is easy to publish. *Publication* is defined under Section 101 of the 1976 Act as "distribution [or offering for such distribution] to the public by sale, . . . rental, lease or lending." This definition is broad and can be applied to the widespread licensing of software. However, a "limited publication" is not "publication." A limited publication is "distribution to a definitely selected group for a limited purpose without right of defusion, reproduction, distribution or sale."[124] This is a troublesome definition for software owners if the "definitely selected group" requirement means more than that the software is distributed only to owners of certain computers who agree to certain restrictions.[125] Arguably, making software available to anyone (with an appropriate

[122] 1909 Copyright Act §§ 2, 24.
[123] 17 U.S.C. § 102(a).
[124] White v. Kimmel, 193 F.2d 244, 746–47 (9th Cir. 1952).
[125] *E.g.*, M. Bryce & Assoc., Inc. v. Gladstone, 107 Wis. 2d 241, 319 N.W.2d 907, 914 (Wis.

computer system) and then restricting use does not meet the definite group requirement. This concept of limited publication is critical to determining whether the distribution of software with restriction licenses is a publication or not.

The legislative history indicates that distributing *to the public* means distributing "generally to persons under no explicit or implicit restrictions with respect to disclosure of contents."[126] This legislative history endorses a favorable interpretation of a limited publication test, to the effect that the test does not mean that dissemination must be to a limited group whose members are ascertained prior to distribution, but only that the group must be restricted from recirculating the work.[127] Case law is also favorable, but not clear. In *GCA Corp. v. Chance,*[128] distribution of object form only to purchasers of another product and for a limited purpose with the belief that it would not be copied was held to be a limited publication.

The distinction between a "limited" or "general" publication first arose under the 1909 Copyright Act when the underlying policies were different than under the 1976 Act. In effect, there were two standards of "publication": an easy one to meet when the just result was to apply federal protection, and a hard one when to apply federal protection would cause the work to be forfeited to the public domain.[129] The "limited publication" doctrine arose to prevent such forfeiture following distribution without a statutory notice. Neither standard is pertinent under the 1976 Act because federal protection applies regardless of publication and an author can take steps (such as registration) to prevent forfeiture even after publication without notice.

Therefore, it is unclear how to apply precedent under the 1909 Act regarding publication and whether the "limited publication" doctrine is still viable. The main effect of publication and the use of statutory notice, as discussed below, is whether trade secret protection is lost either by public disclosure due to publication or election of protection by use of the notice.[130] When the trade secret issues are clarified, these publication issues will be unimportant.

App. 1982) (manuals and forms on how to design certain software systems published where author did not restrict customer copying and redistribution of them); H. W. Wilson Co. v. National Library Serv. Co., 402 F. Supp. 456 (S.D.N.Y. 1975) ("Reader's Guide" held to be published even though it was always provided with a restrictive license, since the persons who could use it were not restricted, only its resale or transfer). *See* American Visuals Corp. v. Holland, 239 F.2d 740 (2d Cir. 1956).

[126]H.R. REP. No. 94-1476, 94th Cong. 2d Sess. 1 at 138, *reprinted in* [1976] U.S. CODE CONG. & AD. NEWS 5569 at 5754.

[127]*See* Public Affairs v. Rickover, 284 F.2d 262, 273 n.2 (D.C. Cir. 1960) (dissent) (that the test does not mean that dissemination must be to an ascertained group, but that the group must be restricted from recirculating the work).

[128]1982 Copr. L. Dec. ¶ 25, 464 (N.D. Cal., Jul. 12, 1982); *see* Hubco Data Prod. Corp. v. MAI, No. 81-1295 (D. Ida., Feb. 3, 1983) (slip opinion at 12–13).

[129]M. Bryce & Assoc., Inc. v. Gladstone, 107 Wis. 2d 241, 319 N.W.2d 907 (Wis. App. 1982).

[130]*See* Part 1b under "Trade Secrets."

b. *Notice on Unpublished Works*

Since it is easy to publish, should the notice be used even though the author believes the work to be unpublished? This question is critical to software vendors who rely upon restrictive licenses and trade secret protection in marketing software, and who want to retain the argument that the work is not "disclosed" under trade secret law. They fear that a finding of copyright publication, although not conclusive, would weaken trade secret protection.

The use of the statutory notice for published work may create an argument that because of the copyright notice the vendor is admitting that the work is being published. Indeed, the ABA Section on Patent, Trademarks and Copyright Law has gone so far as to suggest using a nonstatutory notice on software to avoid this argument. The notice reads: "Unpublished—all rights reserved under the Copyright laws." This position, however, is untenable. If the work were held to be published, since it would not have a proper notice, certain additional steps would have to have been taken to preserve copyright protection, such as registration, which creates other problems.[131] If the work were held to be unpublished, the notice is unnecessary. Rather, a statutory notice which includes the phrase "an unpublished work" may be more prudent, such as the following: "Copyright 1982, an unpublished work by Software Company. All rights reserved." By including the phrase "an unpublished work," the software company is protecting itself in the event the software is determined to have been published.

This problem is not unique to software. Unnamed "professional sports leagues" requested a regulation from the Copyright Office recently that the use of the notice not be construed as evidence of or an admission of publication. The Copyright Office refused, claiming such a regulation was outside its authority.[132]

c. *Date on the Notice*

The statutory notice must include the name or a recognized abbreviation of the name of the owner; the date of first publication; and the appropriate copyright symbol or words.[133] An interesting problem with software is which date to use. Software normally is continually modified throughout its life. Initially, it is subject to extensive testing and modification. Thereafter, it is subject to improvement and the infrequent correction of major defects or "bugs." In some cases, particularly when the software is widely distributed through distribution companies, as is now becoming common in the personal computer market, the date of first publication is relatively clear. In many cases, however, when the software is distributed by restrictive licenses and publication may not have occurred, it is less clear which date to use. The 1976 Act indicates that a date is defective only if it is more than one year after first publication, and is not defec-

[131]*See* Part 2 under "Trade Secrets."
[132]46 Fed. Reg. 58308 (Dec. 1, 1981).
[133]17 U.S.C. § 401(b).

tive if it is earlier than publication.[134] Therefore, safe advice is to use the earliest conceivable date. While one's protection period commences at the early date picked, the period is over fifty years,[135] much longer than the useful life of any now-conceivable program.

The uncertainty with the coverage of binary form has created a problem with an early date, however. The early date may be construed as an admission of publication at that time.[136] If the program was marketed then without an effective notice, it may be in the public domain. This problem is insurmountable only if that date was prior to January 1,1978, when the 1976 Act took effect.[137] Under the 1976 Act, software companies can take certain steps, such as registration, to preserve their rights even though they may have published software without the statutory notice.[138]

A different problem arises if the date of the notice and the date of first publication is between 1978 and 1980. It is possible to argue that binary form was not copyrightable until the 1980 amendments became effective. This argument has constitutional dimensions, particularly if the authority for covering binary form is found to derive from the Commerce Clause, rather than the Copyright Clause; a distinction would be made between the 1976 Act, passed under the Copyright Clause, and the 1980 amendments, passed under the Commerce Clause. Binary form was not addressed until the 1980 amendments. Indeed, the 1976 Act specifically delayed acting on software. The original Section 117 prior to amendment in 1980 expressly continued the law applicable to software under the 1909 Copyright Act and state common law copyright. If between January 1, 1978 (when the 1976 Act took effect), and December 12, 1980 (when the 1980 amendments took effect), binary form was not copyrightable subject matter, if protectable at all, it would have been protected by common law copyright or other schemes of protection.

The uncertainty between 1978 and 1980 brings into question the "backdating" of the notice prior to 1981. Is the notice defective if the subject matter is not copyrightable? If the notice constitutes publication, has publication occurred at a time when the notice cannot help preserve federal protection? The software company would have to argue that common law copyright still applied (or trade secret law) and that distribution of the software then without a notice and now with a back-dated notice does not constitute "publication" at the earlier time divesting the work of common law copyright protection.[139]

---

[134]17 U.S.C. § 406(b).

[135]17 U.S.C. § 302.

[136]*E.g.*, Management Science Am., Inc. v. Cyborg Systems, Inc., 6 COMP. L. SERV. REP. 921 (N.D. Ill. 1978).

[137]17 U.S.C., ch. 3.

[138]17 U.S.C. § 405(a).

[139]Rossette v. Rainbo Record Mfr. Corp., 354 F. Supp. 1183 (S.D.N.Y. 1973), *aff'd*, 546 F.2d 461 (2d Cir. 1976) (held that distribution of musical recordings did not constitute "publication" ending common law copyright protection because musical recordings were then not subject to federal copyright laws). *But see* Data Cash Systems, Inc. v. JS&A Group, Inc., 628 F.2d 1038 (7th Cir. 1980) (under 1909 Act, object code on ROM unprotected when distributed without notice).

If the software has undergone substantial revisions since the early date of first distribution, an alternative is to put several dates on the notice: the date of first distribution and the dates of recent distribution of substantially changed versions (preferably at least the first such date after December 12, 1980). Then, if for any reason the earlier version were held to be in the public domain because of the defective notice or publication without notice, the revisions would still be protectable.

d. *Placement of the Notice*

The Copyright Office has adopted a final regulation for placement of the notice on software.[140] It provides four examples of placement on machine-readable works:

1. Placed such that the notice would appear either with or near the title, or at the end, of printouts;
2. Displayed on a terminal at sign-on;
3. Continuously displayed on a terminal; or
4. Legible on the work's "permanent" container.

Placement in other fashions can still provide requisite notice provided that the notice is "permanently legible" to the user under normal use.[141] Interestingly, the four examples of placement do not necessarily meet the "permanently legible" requirement. For example, a program once loaded in the computer may be "signed-on" continuously, and its users would never see a notice on the original permanent container of the program or displayed at sign-on, nor would they necessarily see it on printouts. Thus, deviation from the four examples may be strictly construed.

On ROMs, the notice should be placed on the container of the chip.[142] The notice can also be placed inside the container on the chip itself (microscopically) and, if several chips together create the copyrighted work, it may be prudent to place the notice both on all chips and on the circuit board near the chips.

For videogames, the notice of the audiovisual work in the game (a different work than the program) can be placed with the title or credits of the work in its "attract mode," at the beginning or end of execution of the "play mode" or on the housing or container of the game.[143]

e. *Use of "(c)"*

The 1976 Act provides for only three symbols for the copyright notice: "copyright," "Copr." and "©."[144] However, the forty-eight-character print matrix of most word processors does not include the copyright symbol ©. Many

[140] 46 Fed. Reg. 58307 (Dec. 1, 1982), which amends the existing regulation at 37 C.F.R. § 201.20. New § 201.20(g) gives four examples of placement on machine-readable works.
[141] 37 C.F.R. § 201.20(c)(1).
[142] 37 C.F.R. § 201.20(g)(4).
[143] 37 C.F.R. § 201.20(h).
[144] 17 U.S.C. § 401(b).

authors now use "(c)" instead. The Copyright Office was asked to sanction such usage, but refused, claiming no authority.[145]

The main reason to use © is for international protection under the Universal Copyright Convention. Although it is doubtful that "(c)" will be found to meet this requirement, at least an equitable or inquiry-notice argument is preserved with "(c)" in lieu of nothing. However, the international requirement is only a substitute for formalities, such as registration. Primarily the only country which requires formalities for which the c-in-a-circle is necessary is the United States. Therefore, the problem of print matrix not having this symbol is mostly a problem facing foreign, not American, authors. Still, the addition of "(c)" would be of great convenience.

TRADE SECRETS

Trade secret protection has been the primary method of protection of software. Many articles or reports discussing the protection of software, including this one, commence with the discussion of patent and copyright problems. The majority of the literature in this area have concentrated on such problems. This emphasis may give the misleading impression that it is for patent and copyright systems that most program owners yearn. Instead, except in the now-developing market for microcomputer software such as videogames, which are distributed much as any consumer entertainment item has been distributed in the past, most software has been well protected by restrictive trade secret licenses.

1. Subject Matter

Trade secrets have been defined to cover any confidential formula, pattern, device or compilation of information which is used in one's business and which gives one an opportunity to obtain an advantage over competitors who do not know or use it.[146] Trade secret law operates differently from patent or copyright law, and instead of encouraging disclosure and invention, its purpose is to maintain standards of commercial ethics as well as to encourage the invention of competitive advantage.[147] The ethics in question oblige persons entering into commercial relationships requiring confidentiality to respect the confidentiality.

Trade secret protection in many ways is preferable to copyright or patent protection. It has been suggested that it is easier to obtain a preliminary injunction, or at least that they are issued more frequently, in trade secret actions than in patent or copyright actions.[148] Also, the secrecy aids protection by preventing

[145]46 Fed. Reg. 58310 (Dec. 1, 1981).
[146]RESTATEMENT OF TORTS, § 757, Comment B.
[147]Kewanee Oil Co. v. Bicron Corp., 416 U.S. 470 (1974).
[148]Bender, *Trade Secret Software Protection,* 3 COMP. L. SERV. § 4–4, art. 2, p. 23.

access and the opportunity to copy. If the software involved is marketed in low volumes and is expensive, infringement may be hard to discover and copyright or other disclosure laws may be too risky to rely upon. Trade secret protection, however, will not be available to a mass distributor of software, because the sale of thousands of copies will negate a claim that the software is secret and will make it impractical to create or enforce restrictive licenses with software buyers.

a. *Trade Secrets in Software*

Software which contains novel elements, such as allowing new applications to be processed by computers, fairly clearly falls within the standard definition of a trade secret. Most software, however, does not contain novelty in this sense, but simply is a different way of repeating the same standard accounting or other applications that comprise the vast bulk of work that computers do. Nevertheless, if the software contains a little-known combination of well-known features performing ordinary functions but which give a competitive advantage, it is protectable.[149]

This combination of features may be unique in every complicated program. Every programmer has pet quirks and every program reflects individual decisions of structure, logic and coding. As a result, programs executing the same transactions will vary in speed, accuracy, cost, flexibility, ease of use, and, above all, commercial feasibility. These inherent differences by themselves have been found to create a ''unique logic and coherence'' which is sufficient to make programs protectable as trade secrets.[150]

Arguably, the mere fact of having a working program, giving one a ''head start'' over competitors working in the same area, is protectable. The head start is created by the dedication of time and effort in the development and programming of the software, even if the ideas and algorithms are commonplace. Programs are not easy to design and write. They also are not easy to debug and make marketable. The very fact that a program works, or that certain design steps were taken which led to blind alleys but later after the expenditure of much time and effort led to a working model, may be protectable. Usually injunctions will issue for trade secrets for the length of time it is estimated to take someone not having access to the trade secret to recreate the same program.[151] Thus, if an employee has had exposure to a project and has seen the blind alleys, and then leaves and pursues the same line of programming, avoiding the blind alleys, it is conceivable an injunction would still issue to preclude the new program from being marketed until as much time passes as it would have taken someone to go down the blind alleys.

[149]Telex v. IBM, 367 F. Supp. 258, 323 (N.D. Okla. 1973), *aff'd on trade secret issue,* 510 F.2d 894 (10th Cir.), *cert. dismissed,* 423 U.S. 802 (1975).

[150]Com-Share, Inc. v. Computer Complex, Inc. 338 F. Supp. 1229 (E.D. Mich. 1971); *see* Cybertek Computer Prod., Inc., v. Whitefield, 203 U.S.P.Q. 1020, 1022 (Cal. Super. Ct. 1977).

[151]*E.g.*, Winston Research Corp. v. Minnesota Mining & Mfg., 350 F.2d 134, 142 (9th Cir. 1965); Analogic Corp. v. Data Translation, Inc., 358 N.E. 218, 804, 807 (Mass. 1976).

b. *The Public Disclosure Problem*

It is a common misunderstanding that trade secrets are easy to lose. Trade secrets are protectable against tortious conduct. Even if the secret was not airtight, if the defendant acted badly in acquiring it, the defendant can lose.[152] The secrecy is lost either by widespread disclosure as opposed to technical slipups or by not taking reasonable precautions to avoid disclosure.[153] For example, Data General Corporation at one time had distributed over eighty copies of a maintenance manual which contained trade secrets regarding the construction of its computers and which were seen or could have been seen by over 6,000 persons. It was able to enforce its trade secret rights despite the broad potential dissemination because it had taken steps to preserve its rights, such as placing proprietary legends on the documents and causing employees and customers to agree to confidentiality restrictions.[154]

Software must nevertheless be kept relatively secret to be protectable. The software company should first institute procedures and take reasonable steps to protect the secrets at the site of creation and marketing.[155] Standard procedures would include restrictive licenses with users; nondisclosure agreements with employees; proprietary and confidentiality legends on all materials embodying the secrets; and restricted access to locations containing the materials or computer systems containing the software, such as by locking doors and cabinets in the evening, having a sign-in, sign-out procedure for the removal of any materials, escorting any guests when in the installation, and having password protection on the computer system.

In this regard, employee nondisclosure agreements can be especially helpful to prevent the theft of source code and other source materials. They may also create rights against the theft of information which falls short of being a trade secret.[156] They can be coupled with a covenant not to compete in many states, notably not including California.[157] If a nondisclosure agreement is not strictly with an employee but with a consultant or independent software house, it should provide that the software being developed is a "work made for hire" under Section 201(b) of the 1976 Copyright Act. Otherwise, the copyright might be owned by the developer. To transfer it would require a written instrument which should be registered as suggested by Section 205. Also, the transfer is reclaimable after thirty-five years pursuant to Section 203.

[152] R. Milgrim, Trade Secrets § 7.08[2].

[153] *Id; See* Motorola, Inc. v. Fairchild Camera and Instrument Corp., 366 F. Supp. 1173, 1186 (D. Ariz. 1973) (no trade secrets when company gave guided tours through sensitive areas without restrictions).

[154] Data General Corp. v. Digital Computer Controls, Inc., 357 A.2d 105, 108, 110–11 (Del. Ch. 1975).

[155] Motorola, Inc. v. Fairchild Camera and Instrument Corp., 366 F. Supp. 1173, 1186 (D. Ariz. 1973) (no trade secrets when company gave guided tours through sensitive areas without restrictions).

[156] R. Milgrim, Trade Secrets § 3.05[1][a].

[157] Calif. Bus. & Prof. Code § 16600.

Disclosure to nonemployees is permissible pursuant to a confidentiality relationship, whether explicitly made confidential by contract or impliedly made confidential by the circumstance of the relationship.[158] The party learning of the trade secret pursuant to the confidential relationship has an obligation not to disclose the secret. Rather than rely upon implied rights, software houses should execute restrictive licenses with users, specifying restrictions which perfect trade secret rights. This procedure is useful, and perhaps essential, to create trade secret rights: the fact that you protect something indicates its importance, and one way of creating trade secret protection is to act as if what is being protected is protectable. Common restrictions include a statement that the software is held in trust and is to be kept confidential; limiting access to it on a need-to-know basis; agreeing to take reasonable steps to prevent unauthorized use, reproduction transfer, or disclosure of it; stating that the software remains the licensor's property; requiring an acknowledgment that the software is or contains trade secrets of the licensor; restricting use to only one computer at a time; requiring all copies to contain the proper proprietary notices and legends; requiring return of all materials and destruction of all computer records when the license is terminated; and being able to terminate the license for any default of the licensee.

In most software licensing arrangements, the party will have access to the materials containing the trade secret (the software) but will not actually learn of any of the trade secrets. Instead, the obligation will be not to attempt to learn the trade secrets. This differentiates software licenses from standard trade secret licenses. Usually trade secret licensing involves the passage of know-how for the creation of a manufacturing process, and inherent in that transfer is the disclosure of the trade secret. Because the person learning of the trade secret may be concerned that it really is not a secret, standard trade secret licenses usually contain extensive exceptions for information in the public domain. In the software context, with respect to the software itself, unless the source code is supplied, these exceptions are not applicable. The user is not supposed to have any knowledge of the trade secret information and would be largely unable to determine whether or not the information is in the public domain. However, many trade secret licenses purport to cover manuals and other documentation in addition to the software, and with respect to such manuals and documents, the standard exceptions for public domain information are more appropriate.

c. *Copyright Compared*

Copyright protects the specific logic and design of the program. Trade secret law protects the unique logic and coherence of the program as well as the underlying programming techniques, routines and algorithms of the program, the input and output formats of the program and the ways in which the program

[158]Data General Corp. v. Digital Computer Controls, Inc., 357 A.2d 105, 110–11 (Del. Ch. 1975).

interfaces with other parts of the computer or the surrounding environment. The fundamental advantage of trade secret law is that it protects more than the logic and design of the program itself.

Even with respect to the logic and design of the program, trade secret law has an advantage over copyright. The test for infringement is different; theoretically, in a trade secret case, even if source code is modified to look not at all "substantially similar" it could still be protectable. Trade secret law would still apply despite extensive changes for any one of the main reasons for its applicability to software. The novel ideas in the original source code may not have been changed; a particular advantageous combination of features which made the original program marketable may have been appropriated; the "unique logic and coherence" of the original software may still be for the most part retained; and, fundamentally, the head start of the original owner has probably been damaged since it would undoubtedly take the infringer less time to do the changes than it would have taken the infringer independently to develop the software.

Over time, the advantage of using trade secret law to protect the program itself may become less apparent. Often the proof in such trade secret cases will be similar to that in copyright cases and will revolve around certain "fingerprints" in the original source code (such as the programmer's initials or a programmer's pattern of misspelling) which by oversight reappear in the changed version.[159] In addition, the common law development of software copyrights may extend protection beyond that normally given copyrighted expression, and eventually copyright protection may be found to extend almost as far as would trade secret protection.

Even then, trade secret protection will still retain certain advantages. It can inhibit (by restrictive licenses) public disclosure of the software and thus limit the number of persons who might be possible infringers and lessen the likelihood of infringement. It is also applied fairly uniformly around the country and has less unresolved issues currently than copyright has respecting software, including clear applicability to object code; coverage of intermediate materials, such as flow charts, ROMs, chip masks, and diagrammatic source materials; and (as discussed below) the availability of international coverage.

Trade secret law has been criticized by advocates of copyright or patent schemes for software on the grounds that it is easy to lose (since once the secret is no longer a secret, it cannot be protected as a trade secret). This assertion is moderately misleading, for technical disclosures of trade secrets have not been sufficient to relinquish the secrets to the public domain and diligent software vendors should be able to put a cap on such a release by vigorous litigation.[160]

Trade secret law, however, gives less protection than copyrights or patents

[159]In Structural Dynamics Corp. v. Engineering Mechanics Mechanisms Research Corp., 401 F. Supp. 1103, 1117 (E. D. Mich. 1979) and in Williams Elec., Inc. v. Artic Int'l, Inc., 685 F.2d 870 (3d Cir., Aug. 3, 1982), copying was shown by such "fingerprints."

[160]R. Milgrim, Trade Secrets, § 7.08[2].

against third parties. The third party would have to obtain the trade secret wrongfully through a licensee or employee (in which case that licensee or employee could be sued), or the third party would have to have had knowledge or reason to believe that it was obtaining a trade secret (in which case the third party could be sued).[161] Still, the problem of third parties may be more theoretical than actual. The trade secret owner is most concerned not when it loses royalties because of licensees cheating on a license or giving copies to friends, but when a concerted effort is made to create a competing business. In that circumstance, it is generally essential for the competing business to have access to the source materials. Normally, this cannot be done without violation of trade secret law because a trade secret owner generally only markets object code.

2. The Copyright Registration Problem

The reliance on trade secret protection in addition to copyright law may not be effective because of problems caused by copyright registration. Under the 1976 Act, registration is not required for copyright protection but it does have certain advantages. A Certificate of Registration is *prima facie* evidence of ownership and validity of the copyright.[162] Registration is a prerequisite to bringing an infringement action.[163] Registration entitles a copyright owner to statutory damages and attorneys' fees against infringements occurring after registration.[164] Registration can also cure defective notice and avoid release of a copyrighted work into the public domain.[165]

a. *Disclosure of Trade Secrets*

Registration, however, may result in disclosure of trade secrets. Normally, registered works are deposited both in the Library of Congress and in the Copyright Office, and in both cases are available for public inspection (although copyrighted works inspected at the Copyright Office may not be copied without permission of the copyright owner).[166] The disclosure of trade secrets can result from several situations.

Public inspection could result in disclosure of trade secrets. As a practical matter it may be difficult to inspect the program without copying it, particularly since it is not allowable that a physical copy be made and taken for more careful scrutiny outside the Library of Congress or the Copyright Office. Still, allowing such inspection makes protection of trade secrets questionable. Only

[161] *Id.*, § 5.04. *See* Computer Printing Systems, Inc. v. Lewis, 212 U.S.P.Q. 626 (Pa. Super. Ct. 1980) (licensee "innocently" received source code from disloyal officer of licensor but was nevertheless liable for unjust enrichment in using the source code for creating competing software).

[162] 17 U.S.C. § 410.

[163] 17 U.S.C. § 411.

[164] 17 U.S.C. § 412.

[165] 17 U.S.C. § 405–46.

[166] 17 U.S.C. § 407–08. The Library of Congress requirements are identical to the Copyright Office rules; *see* 37 C.F.R. 202.19(d)(1).

certain parts of the work may need to be seen to ascertain the critical secrets; more to the point, in convincing a judge to issue a preliminary injunction, the fact that software is available for inspection may make later success on the merits questionable unless it was not in fact inspected or if no secrets could be revealed when it was inspected.[167]

The very deposit of the work, particularly with the Library of Congress where it will be indexed and potentially exposed to widespread inspection, is hardly an act which retains the "secrecy" of a trade secret. Deposit may be deemed as a matter of law to be an act inconsistent with treating the work as secret. Trade secret protection would then not exist even if the trade secret infringer in a particular case did not in fact attempt to inspect the work where deposited.

Even if a registered copy of software is still somehow "secret" or if a system of registration in secrecy is created (as discussed below), its secrecy conceivably could later be lost by release due to a Freedom of Information Act (FOIA) request. The Copyright Office is subject to the FOIA, at least to the extent of "actions taken by the Register of Copyrights. . . ."[168] Arguably, the Register in accepting deposit of registered works is taking an action which falls within the scope of the FOIA. Since deposited works are open for public inspection regardless of the FOIA, this problem is largely academic. It is not surprising that the regulations of the Copyright Office implementing the FOIA currently cover only the records respecting deposited works, but not the works themselves.[169] Now that certain tests such as the SAT are allowed secure deposit, this previously academic issue may soon arise.[170]

The FOIA provides for several exemptions which may be applicable. The FOIA allows an agency not to disclose a "trade secret."[171] It is not yet clear, however, how the Copyright Office would interpret what the FOIA means by *trade secret.* The Copyright Office would make its own decision as to what constitutes a trade secret.[172] The Copyright Office can consider certain public interest issues, such as whether retaining the secrecy may harm the public;[173]

[167]*Cp.* Franke v. Wiltscheck, 209 F.2d 493 (2d Cir. 1953) *with* Carson Prod. Co. v. Califano, 594 F.2d 453 (5th Cir. 1979).

[168]17 U.S.C. § 701(d). The extent of the Copyright Office's obligations under FOIA are as specified under § 701(d). Without it, the Copyright Office would be exempt since it is a part of the Library of Congress, which is part Congress, and "Congress" is exempt from the FOIA under 5 U.S.C. § 551(1)(A). *See* Reporters Comm. for Freedom of Press v. Vance, 442 F. Supp. 383, 385 n.5 (D.D.C. 1977), *aff'd,* 589 F.2d 1116 D.C. Cir. 1978), *aff'd in part rev'd in part sub nom.* Kissinger v. Reporter's Comm. for Freedom of Press, 445 U.S. 136 (1980).

[169]37 C.F.R. § 203.4.

[170]The issue was not raised in *National Conf. of Bar Examiners v. Multistate Legal Studies,* 692 F.2d 478 (7th Cir., Nov. 2, 1982) (secure deposit of Multistate Bar Exam). Related issues under a state equivalent to the FOIA were raised in *Associates of Am. Med. Coll. v. Carey,* 482 F. Supp. 1358 (N.D.N.Y. 1980) ("Truth in Testing" release of MCAT).

[171]5 U.S.C. § 552(b)(4).

[172]F.T.C. v. Owens-Corning Fiberglass Corp. 626 F.2d 966 (D.C. Cir. 1980).

[173]In making the decision to disclose or withhold a trade secret, the agency can legitimately consider factors such as whether disclosure would significantly aid the agency in performing its

since the Copyright Office has generally been opposed to the registration of software in secret, a program owner might be ill-advised to risk the Copyright Office's weighing of such factors. The FOIA also prevents disclosure of items "*specifically* exempted from disclosure by statute" (emphasis added).[174] The 1976 Copyright Act has been interpreted to allow the Copyright Office to issue regulations restricting disclosure.[175] To be effective against the FOIA, however, the Copyright Act may need to be amended to provide clear statutory authority.[176]

Ironically, the software owner's best protection against the FOIA is the owner's statutory copyright itself. The Copyright Office is arguably statutorily precluded from producing a copy in compliance with an FOIA request without permission of the copyright owner (assuming such copying would not be considered fair use).[177]

b. *Techniques to Retain Secrecy*

Prudent advice is to avoid the possibility of losing trade secret protection by not registering software if that can be avoided. A system of registration of software in secret, not subject to the FOIA, would be very helpful to the software industry. In the meantime, a variety of techniques have been proposed or are available to retain secrecy.

IDENTIFYING MATERIALS

The most commonly used method to retain secrecy is to deposit "identifying materials" instead of the complete program. By a specific regulation relating to software, the Copyright Office exempts from deposit in the Library of Congress computer programs published in the United States in machine-readable form only.[178] This regulation covers a substantial number of software houses which only market their software as object code in binary form, although many of these houses would take the position that they have not "published" their works and perhaps do not fall precisely within the regulation.

The regulation also exempts deposit of the complete program in the Copyright Office, and instead requires deposit of "identifying materials" consisting

functions, the extent it would cause harm to its procedures and to the public, and alternatives which would serve the public interest. Doctors Hospital of Sarasota, Inc. v. Califano, 455 F. Supp. 476 (1978).

[174]5 U.S.C. § 552(b)(3).

[175]National Conf. of Bar Examiners v. Multistate Legal Studies, 692 F.2d 478 (7th Cir., Nov. 2, 1982) (finding secure deposit regulation for the "Multistate Bar Examination" consistent with the Copyright Act under 17 U.S.C. § 408(c)).

[176]*E.g.*, Irons & Sears v. Dann, 606 F.2d 1215 (D.C.Cir), *cert. denied*, 444 U.S. 1075 (1975) (FOIA request for undisclosed, abandoned patent applications denied due to specific exemption).

[177]*Cp.* 17 U.S.C. § 706(b) with Weisberg v. Department of Justice, 631 F.2d 824 (D.C. Cir. 1980) (FOIA request for copyrighted photographs of *Life* magazine held by FBI not granted because the copyright owner had not been joined as defending or given consent). *But see* 37 C.F.R. § 203.5(a), allowing a copy to be made of materials requested under the FOIA.

[178]37 C.F.R. § 202.19(c)(5) exempts from the Library of Congress' deposit requirements computer programs published in the United States in machine-readable form only, from which the work cannot ordinarily be visually perceived except with the aid of a machine or device.

of the first and last twenty-five pages of the program and the page containing the copyright notice. The materials deposited must be in human-perceptible form.[179] Typically, the first and last twenty-five pages of the source code would be reorganized before deposit such that these pages contain only irrelevant "filler" or nonsensitive information so as not to compromise trade secrets. Some programs are too short or have insufficient modularity to be reorganized, however. Also, blatant reorganization may allow an infringer to question the validity of registration, possibly delaying issuance of a preliminary injunction.

The Copyright Office may show flexibility on the issue of deposit of identifying materials, and may accept other than the first and last twenty-five pages, such as every other page or some other structure which indicates authorship but which does not reveal sensitive matters. Also, conceivably a program could be identified by a videotape of its output. In such a registration, however, the copyright owner should make it very clear that the underlying program is being registered, not the audiovisual work represented by the output.

DEPOSIT OBJECT CODE

A second approach to retain secrecy is to deposit object code rather than source code. Theoretically, trade secrets are then not being compromised. Many of the trade secrets may not be ascertainable from object code but would require access to the source code. The object code cannot be copied out of the Copyright Office or decompiled. In addition, if supplied in binary form, not a printout, and in a format for which no translating machine exists at the Copyright Office, it could not even be perceived by any would-be inspector.

The main impediment to this approach is the Copyright Office, which would rather have source code. It claims that it is difficult to determine authorship from binary form or object code.[180] This view of the Copyright Office is not consistent with its acceptance of foreign language works, encrypted works, or works in non-Latin script in the sense that it is no easier to determine authorship from them than it is from object code. Still, the 1976 Act requires that the "best edition" of the work be deposited,[181] which the Copyright Office interprets to mean the best representation of authorship: a source code printout. Even then, if object code is determined not to be a "copy" but a "derivative work" of source code, it needs to be separately registered. Object code is also the version that is most often being distributed and the one that needs that protection from copiers.

The Copyright Office will accept object code deposits, but only after sending a "rule-of-doubt" letter.[182] The rule of doubt is the policy of resolving

[179]The program must be "reproduced in a form visually perceptible without the aid of a machine or device." 37 C.F.R. § 202.20(c)(2)(vii).

[180]1981 CLA Transcript at 196.

[181]17 U.S.C.. § 408(b)(2).

[182]The letter is designated as GLR-70. The doubt stated is not that the program is not within the subject matter of copyright, but that since the Copyright Office examiners are not expert computer programmers, they are unable to determine that the object code deposited actually contains copyrightable authorship. 1981 CLA Transcript at 129. The letter reads in part as follows:

doubtful issues in favor of registration. The effect of this rule is not clear. The rule of doubt may be found to be outside the Copyright Office's discretion. Section 410 gives the Office only two options: to accept or reject registration. If the rule of doubt is *ultra vires,* it may be ignored by a court—or it may be found to constitute a rejection of registration, possibly seriously delaying or damaging a software owner's action for infringement. Even if it does not remove the advantages of registration in establishing a *prima facie* case, it may make rebuttal of that case much easier—which makes it more difficult to obtain preliminary injunctions in copyright infringement actions.[183]

SPECIAL RELIEF

A third route for retaining secrecy is to apply for "special relief" under the Copyright Office regulations.[184] This means that the source code would be inspected for evidence of authorship and then returned except for minimal identifying portions. The Copyright Office has used it for such tests as the SAT, but does not officially sanction this approach for software, except on an *ad hoc* basis.[185] For a particularly sensitive program, it may be prudent to apply for special relief before relying upon depositing object code and being subject to the rule of doubt or depositing identifying materials and having to restructure the program. The application for relief may need to be creative; the Copyright Office will want to retain some type of identifying materials. One idea: photograph source code through diagonal slits, so only slices of code are visible; the original code can still be identified from these slits, but no intelligence or secrets would be revealed.

TRADE SECRET ACT

Another approach which has been suggested is based upon the Trade Secret Act,[186] which provides that a public employee who discloses trade secrets entrusted with his governmental agency can be fined and dismissed. While

---

Dear Remitter:

We are delaying registration of the claim to copyright in this work because the deposit consists of a printout of the computer program in object code or other non-source code format.

The Copyright Office generally requires the best representation of the authorship for which copyright is being claimed. Because Copyright Examiners are not skilled computer programmers, they have extreme difficulty in examining computer programs in other than source code format to determine whether the deposit contains copyrightable authorship. . . .

The Office believes that the best representation of the authorship in a computer program is a printout of the program in source code format. Where, however, the applicant is unable or unwilling to deposit a printout in source format, we will proceed with registration under our "rule of doubt," upon receipt of a letter from the applicant assuring us that the work as deposited [i.e., in nonsource code format] contains copyrightable authorship.

[183]1981 CLA Transcript at 196–97.

[184]37 C.F.R. § 202.20 (d); *Cp.* 37 C.F.R. § 202.19(e)(1), regarding deposit in the Library of Congress.

[185]37 C.F.R. § 202.20(c)(vi). A request for treatment of software like secure tests was denied. 43 Fed. Reg. 772 (Jun. 4, 1978).

[186]18 U.S.C. § 1905.

there is no private cause of action for this, or for an injunction based upon it, the action of an agency on disclosing trade secrets of a business is reviewable under the Administrative Procedures Act.[187] If one is denied special relief, one could then seek review under the Administrative Procedures Act for violation of the Trade Secret Act. The denial should then be sanctioned under the Trade Secret Act only if it is ''authorized by law.''[188] Regulations can supply this authority if substantive and rooted in a grant of such power by Congress.[189] Section 706 of the 1976 Act, which provides that deposited copyrighted works will be available for inspection, may provide such authorization by law. This authorization is not clear, however, because there has been no indication of particular consideration of trade secret problems in Section 706 and because Section 706(b) allows the Copyright Office to make restrictions on disclosure at its discretion.

SECTION 411(a)

One other approach to retain secrecy has been suggested in connection with applications for special relief.[190] It is based upon Section 411(a) of the 1976 Act, which allows an applicant refused registration after complying with all formalities to institute an infringement action. (Normally, registration is a prerequisite to instituting an action.) The approach is to apply for registration with special relief, and, if special relief is refused, gain return of all deposits. Secrecy is retained. When an infringement action is later instituted, claim jurisdiction despite lack of registration based upon Section 411(a). Section 411(a), however, appears intended to prevent subject matter rejection as opposed to formalistic rejection or rejections of collateral requests, such as for special relief.[191] Arguably, the original application was not in proper form and the rejection was not the type for which Section 411 provides relief.

c. *Policy Considerations*

It is puzzling why the Copyright Office has refused to accept programs as secret or to grant special relief. It may believe that supplying identifying materials is a sufficient compromise, although many programs cannot be restructured to avoid disclosing sensitive parts in the first and last twenty-five pages, and many programs are not distributed either by publication or in machine-readable form only. Perhaps it wants copyright alone to protect software, and does not want to encourage trade secret law, patent law or a mixture of various protective schemes. It may interpret the 1976 Act as requiring at least minimal

[187]Admin. Proc. Act § 10(e)(2)(A). *See* Chrysler Corp. v. Brown, 441 U.S. 281 (1979) (reviewable based upon Trade Secret Act); Burroughs Corp. v. Brown, 501 F. Supp. 375, 383 (E.D. Va. 1980), *rev'd on other grounds,* 654 F.2d 299 (4th Cir. 1981) (reviewable based upon FOIA exemptions).

[188]*Id.*

[189]*See* Chrysler Corp. v. Brown, 441 U.S. 281 (1979) (must show a nexus between the regulations and some delegation of the requisite legislative authority by Congress showing a consideration of trade secret concerns).

[190]1981 CLA Transcript at 91-92.

[191]*Cf.* International Trade Mgmt. v. U.S., 553 F.2d 402 (Ct. Cl., Dec. 17, 1982).

identifying materials, although Section 408(c) gives it broad discretion to exempt works from deposit requirements.

Another reason may be the conflicting policies behind copyright protection. Traditionally, the monopoly of copyright protection was provided to encourage public disclosure. The Copyright Office may still agree. The 1976 Act, however, changes the thrust of copyright from publication to creation, and it sanctions federal protection of unpublished works. The 1976 Act also encourages registration and archival storage. These goals can be met without disclosure. Indeed, the present structure discourages registration, thereby discouraging both archival storage and disclosure.

Perhaps the fundamental reason for the Copyright Office to concentrate on disclosure is the political and economic situation facing the Copyright Office and Congress. Disclosure systems benefit large vendors (which have political clout) because of their ability to enforce their rights and to take advantage of disclosure. Registration inhibits start-up companies (which lack political clout) because of their inability to enforce their rights and the high risk that disclosure will benefit competitors with a much more substantial market position and monetary resources.

d. *Analysis*

Rather than ponder the ways to retain registration secrecy, a more important question is: Why require registration of programs at all? It is specious to view registration as facilitating academic development of the ''art'' of programming. Most programs are mundane. Most developments will be kept secret, for commercial advantage. At present only identifying materials need to be registered and they can be fixed so as not to show any advances in the art. Also, it is fiction to believe that persons operating in the software industry periodically review copyrighted works deposited in the Copyright Office through public inspection in order to learn the new techniques. Published programs, unlike books, are not distributed to be read but to be used.

Registration simply fills warehouses. Commercial programs are continually revised, and requiring all revisions or all major revisions to be deposited serves little purpose. Proof considerations—having a copy which can be presented at a trial to prove which copy was copyrighted—could be important, but has not been expressed as a major reason for deposit. Independent proof copies could be acquired, if necessary, from other customers of the copyright owner. Also, these considerations can be met while still having a deposit in secret. Still, unless every new release of software is deposited, the infringed version may not look substantially like the registered version; instead of facilitating proof, deposited copies may impede such proof.

Therefore, it would be better to have no deposit requirement at all. Registration could be performed by reviewing the submitted work for authorship and returning it, or, at worst, accepting it in secrecy, subject to the future rights of defendants.

3. The Copyright Preemption Problem

a. *Federal Preemption*

Section 301 of the 1976 Act provides that the Act preempts certain types of state law. The 1976 Act extended federal protection to unpublished as well as published works, and was intended to preempt common law copyright. Common law copyright protected personal papers much as a trade secret law protects business secrets.[192] Common law copyright gave persons the right to keep private papers from being published if they were appropriated through tortious conduct, such as breaking and entering into a house or other private areas, or breaching a confidentiality relationship.[193] The language of Section 301 does not directly mention trade secret law or common law copyright, but instead calls for preemption of state law remedies which are "equivalent" to exclusive rights given to the copyright holder and which protect copyrightable subject matter.

Legislative history is favorable to the position that trade secret law has not been preempted. The House, but not the Senate, addressed the issue in the 1980 amendments to the Copyright Act, and in its *Report* the House concluded that no preemption should occur.[194] On the floor of the House at the time of the passage of the 1976 Act, however, an extensive list of types of state laws not to be preempted by Section 301 was removed from the Act by voice vote. This removal was apparently due to the overbreadth and vagueness of one of the listed state laws, i.e., "misappropriation."[195] "Trade secret law" was not in the list. This may imply that it was meant to be preempted. This may also imply the opposite. Although it is similar to some of those laws which were removed, it protects different rights than misappropriation. Therefore, the removal of the list does not speak either way as to preemption of trade secret laws.

The basic argument as to why trade secret law should not be considered preempted by the 1976 Act is that trade secret law protects different subject

---

[192] 1 M. NIMMER, COPYRIGHT § 2.02.

[193] *Id.*

[194] H.R. REP. No. 96-1307, 96th Cong., 2d Sess. 1 at 23-24, *reprinted in* [1980] U.S. CODE CONG. & AD. NEWS 6460 at 6482-83. The *CONTU Report* at 18 also concluded (without analysis) that § 301 did not preempt trade secret law.

[195] Originally, § 301(b)(3) would have had a laundry list of state rights which would not have been preempted including "breaches of contract, breaches of trust, invasion of privacy, defamation and deceptive practices, such as passing off and false representation." Section 301(b)(3), H.R. 4347, 89th Cong., 2d Sess. (1966). Each of these rights protects rights other than mere copying and arguably none of them concern rights equivalent to copyright. The bill sent to the floor of the House contained this laundry list with the addition of some others, including one that resulted in considerable controversy: "misappropriation not equivalent to any of such exclusive rights." Various parties, including the Department of Justice, objected to the exemption of "misappropriation" on the grounds that "misappropriation" was so vague and uncertain that it might effectively nullify preemption. Letter from Michael M. Uhlmann, Assistant Attorney General for Legislative Affairs, Department of Justice, to Congressman Robert Kastenmeier (July 27, 1976). As a result of the opposition against the word *misappropriation*, the entire laundry list was scratched. 122 CONG. REC. H32015 (Sept. 22, 1976).

matter than does copyright law. Copyright protects expression and not ideas; trade secret law protects ideas regardless of the form of expression.[196] For example, if Coca-Cola's famous formula were written down, assuming that this writing would then be copyrightable, would it no longer be protectable as a trade secret?

Still, there is overlap between the expression and ideas in software, and arguably trade secret protection of software is narrowed by Section 301. Trade secret law may no longer protect any copyrightable expression which may exist in software. The expression in software is its specific logic and design; trade secret protection extends to the "unique engineering, logic and coherence" of a program[197]—the same subject matter. Therefore, trade secret protection may be restricted to protecting the underlying algorithms, ideas and programming techniques in software, or the input and output formats and interfaces of software, matters not protected by copyright, and may be preempted from protecting the program itself. To again use the Coca-Cola example, if the secret formula were implemented by programs, trade secret law would still protect the formula but would not apply to the unique way the programs were designed and coded.

Even the "unique logic and coherence" of a program could still be protected by trade secret law if the rights granted by trade secret law are not "equivalent" to the exclusive rights given to a copyright holder. This is the second essential part of the preemption test under Section 301. Trade secret rights may not be equivalent because copyright restricts unauthorized copying while trade secret law restricts unauthorized use or disclosure.[198] This distinction may not apply to programs, however, since new Section 117 discusses its use. Arguably, Congress believed that "equivalent rights" included use of software, and felt it had to make an express exception to those equivalent exclusive rights. Still, trade secret law protects more than merely use. It gives remedies for tortious invasion of privacy and breach of trust or breach of confidence, different rights than copying or use. Avoiding preemption may require that the infringement actions be framed in terms of these other remedies, and as a result this may create a narrower trade secret law than currently is accepted in most states.[199]

Even if Section 301 preempts state trade secret law, there may be constitu-

[196]No preemption: Warrington Assoc. v. Real-Time Eng. Sys., 522 F. Supp. 367, (N.D. Ill. 1981); Bromhall v. Rorvik, 478 F. Supp. 361 (E.D. Pa. 1979); Technicon Medical Info. Sys. Corp. v. Green Bay Packaging, Inc., 1982 Copr. L. Dec. ¶ 25, 438 (7th Cir., Aug. 30, 1982) (dictum); M. Bryce & Assoc., Inc. v. Gladstone, 107 Wis. 2d 241, 319 N.W.2d 907 (Wis. App. 1982) (dictum).

[197]Com-Share, Inc. v. Computer Complex, Inc., 338 F. Supp. 1229, 1234 (E.D. Mich. 1971).

[198]No preemption: Warrington Assoc. v. Real-Time Eng. Sys., 522 F. Supp. 367, (N.D. Ill. 1981); M. Bryce & Assoc., Inc. v. Gladstone, 107 Wis.2d 241, 319 N.W.2d 907 (Wis. App. 1982) (dictum).

[199]Recent case concurs—and found preemption where claims of infringement were not framed as above. Avco Corp. v. Precision Air Parts, 210 U.S.P.Q. 894 (M.D. Ala. 1980), *aff'd on different grounds,* 676 F.2d 494 (11th Cir. 1982).

tional limits on the extent of preemption. Binary form may not be a "writing" of an "author" under the Patent and Copyright Clause. If so, the authority for copyright for binary form may derive from the Commerce Clause and be limited accordingly. The Commerce Clause has been given very broad applicability, however, and would probably cover all procurements of software except perhaps software developed internally and not marketed in interstate commerce. This would not necessarily be a bad result. If trade secret law were held still to apply to enforce employee confidentiality agreements with respect to the development of software, then software houses marketing complex and sensitive software could protect their source materials by such agreements under trade secret law and could market and protect binary form by copyright.

Answering the question of whether Section 301 preempts trade secret law may not be conclusive. Traditional preemption doctrines may be applied by courts to provide for preemption beyond or different from that specified in Section 301. Also, Section 301 may not apply to software for cases arising between 1978 and 1980. Former Section 117 continued the law applicable to software in effect before the 1976 Act took effect, until CONTU had concluded its investigations. Arguably, the standards of Section 301 did not control respecting preemption until December 12, 1980, when Section 117 was amended.

b. *Election of Protection*

Sometimes the preemption issue is confused with election of remedies.[200] Defendants recently have argued that a copyright notice on a program is an *admission* of publication.[201] Publication for copyright purposes, however, is not necessarily the same as disclosure for trade secret purposes. Defendants have also argued that use of a copyright notice may give rise to an *estoppel*, that is, use of one precludes reliance upon the other. In *Technicon Medical Informations Systems Corp. v. Green Bay Packaging, Inc.*,[202] this argument was intensively considered and rejected. Fundamentally, this argument is archaic because the 1976 Act recognizes copyright in unpublished works. No longer must one elect copyright protection by the act of publication; it attaches (whether one wishes it or not) at the moment of creation of a work in a nontransient form.

c. *Analysis*

The policy question is whether copyright law or any other regulatory system protecting software should supplant trade secret protection. The Supreme Court has recently compared the policies of federal patent law with trade secret law and found that in those contexts preemption was not appropriate.[203] From a

[200]*E.g.*, BPI Systems, Inc. v. Leith, 532 F. Supp. 208 (W.D. Tex. 1981) (trade secret law not preempted since "the material improperly used was not copyrighted").

[201]*E.g.*, Management Science Am., Inc. v. Cyborg Sys., Inc., 6 Comp. L. Serv. Rep. 921 (N.D. Ill. 1978) (1909 Act).

[202]1982 Copr. L. Dec. ¶ 25,438 (7th Cir., Aug. 30, 1982) (no estoppel).

[203]Kewanee Oil Co. v. Bicron Corp., 416 U.S. 470 (1974) (federal patent law does not preempt state trade secret law); Aronson v. Quick Point Pencil Co., 440 U.S. 257 (1979) (trade secret license relating to a mechanical structure was upheld).

narrower point of view for this industry, trade secret law should not be preempted because it would change the marketing practices and assumptions of software houses.

The dilemma facing software houses is as follows: An employee who steals source code can modify it so that it is not recognizable. This makes it unlikely that the software house could prove a copyright violation because the source code would no longer look substantially similar. Therefore, the best protection and the traditional protection is to rely upon employee nondisclosure agreements.

If a licensee is given source code, the same problem could occur. With respect to many forms of software, it is possible to market object code, and copyright should work in this context since it is very hard to modify object code or to make it look other than substantially similar. However, if object code is marketed without trade secret protection, that in itself may comprise trade secret protection; in addition certain copyright procedures must be followed, the most important of which is to register the program prior to suing for infringement. Registration, however, as discussed above, may result in the disclosure of the source code version of the program.[204] This may cause loss of the use of trade secret protection of the source code with respect to employees; it may also enable competitors to inspect the registered version of the program and to copy it in a manner which it is difficult for the software company to discover or to prevent.

Further, if manuals are licensed with the subject code, the manuals and the way in which the object code work may give clues to the licensee of how the source code is structured. Copyright does not preclude use of this knowledge to reverse engineer the program.[205] Trade secret protection of the manuals, the object code, input forms, operating procedures, output and data sets created by the program on the computer may preclude use of such clues.

Therefore, trade secret law should not be preempted. Nevertheless, the dilemma of software houses can be solved even if trade secret law is partially preempted as long as trade secret law still restricts disloyal employees and outside consultants who are provided with more than just a program and with manuals. Returning to the Coca-Cola example, it is those who are told the formula who must be restricted, not those who are given programs which implement the formula but from which the formula cannot be ascertained.[206] The programs can

[204]*See* Part 2 under "Trade Secrets."

[205]Synercom Technology, Inc. v. University Computing Co., 462 F. Supp. 1003 (N.D. Tex. 1978).

[206]Interestingly, copyright protection of binary form enhances trade secret protection of underlying algorithms. It is a violation of copyright to print out the code from binary form, or to decompile it if it is object code. Therefore, it is legally impermissible to make binary form intelligible—and therefore impermissible to attempt to reverse engineer the trade secrets which may be in binary form. Thus, distribution of binary form under copyright licenses should not disclose trade secrets in source or object code. For example, if the Coca-Cola formula is embodied in a machine which mixes the necessary ingredients in the right combinations, and the machine contains a program in binary form to do this, binary form which incorporates the trade secret formula but from which one

be adequately protected by copyright. Thus, partial preemption of trade secret law as outlined above—of protecting the unique logic and design of programs against copying or deriving related programs—will not significantly affect the software industry. Software houses could still protect underlying formulas or obtain relief against disloyal employees, since either the subject matter protected would not be copyrightable or the rights being protected would be other than copying rights.

This argument still leaves software houses up in the air. How do they act now to preserve their rights in light of preemption or election of remedies arguments? Relying purely on trade secret protection may result in loss of protection due to the preemption of trade secret law and the resulting problem of not having had a statutory notice for copyright protection. Placing a statutory notice may result in the election of remedies. One answer is to place the words "an unpublished work" with the statutory notice and also to place a trade secret legend on the software. The use of the phrase "an unpublished work" probably avoids the problem of creating an admission of publication; the use of the statutory notice avoids loss to the public domain in the event of preemption; and the use of a trade secret legend argues against an election of protection.

OTHER PROTECTIVE SCHEMES

1. Technological

Inadequacy of legal methods has led to the use of technological self-help techniques. Patent protection has been expensive and unreliable. Copyright protection may not be sufficient and may not apply to all forms of programs. Trade secret protection requires increased costs for internal security procedures and marketing, as many trade secret licenses with users are negotiated, and is potentially impermanent. Even if an enforceable method of protection is discovered, detection of infringement is difficult. Enforcement is expensive and there is always the risk of a capricious decision. Uncertainty in the relation of copyright to trade secret law enhances this risk. Therefore, software houses have searched for self-help remedies. Unfortunately, even self-help methods often fail through the efforts of the idle minds of software hobbyists.

The major self-help mechanism is to distribute object code only. The ideas in the source code are very difficult to discover from object code and as a practical matter it is not always feasible to reverse engineer the source code from the object code. Wang, a word processing company, takes this method much further than usual: it distributes only ROMs which are already in its machines. Currently, one of the main limitations on this method is the comprehensive and detailed operating manuals which often have to be provided with the object code. These manuals are sometimes distributed without consideration of their need to be protected and can provide the basis for reverse engineering.

could not legally ascertain the trade secret formula, in what sense is distributing the binary form disclosing the trade secret?

The second major self-help technique is to rely upon economic leverage. If software is complicated and not fully debugged, customers will eagerly pay for maintenance and will be loathe to violate a license or use a pirated copy and lose access to the maintenance. Indeed, one of the major ways in which software houses discover piracy is when an innocent acquirer of a pirated copy asks for the most recent version of the software because of certain bugs the user has discovered. In addition, software houses often offer updates or new releases to their licensed customers. These updates not only contain corrections of errors but also can contain improvements or changes to bring the software in conformity with changing laws or changing technology. Licensed customers can often obtain the updates at a discount compared to acquiring a new license.

Another common self-help method is "fingerprinting." Fingerprinting is easy to implement and quite prudent in anticipation of litigation. Fingerprinting consists of placing identifying items in the program, including dummy routines with no useful purpose; unnecessary code, such as specifying a constant value for a variable; contorted methods of coding, unlikely to be duplicated independently; initials, names or other identifying symbols in the comments to the code; insignificant mistakes; and a serial number for each copy of the software to be able to trace the source of piracy.

A developing area of self-help is to create techniques which internally limit the ability of a machine to copy the program. One method is for the program to have a timer or otherwise to count the number of times it is operating. This can be done by referring to the internal clock of a computer and checking a start date with the then-current date, or it can be done by counting the number of times a certain operation or combination of operations is executed. The program then can be designed to self-destruct unless it receives certain code words. A lawyer recommending this method be implemented should also consider whether the user should be notified about this self-destruction capability, to avoid a large claim for damages arising from unexpected early self-destruction.

Another internal self-help method is to have the program check the serial number of the computer or another hardware-dependent item in the computer and to become inert if it is on the wrong machine. One problem this creates is how to allow for the use of the software on a back-up computer. One answer is to combine the self-destruction capability with the hardware-dependent or serial number capability. In this context, use on a back-up machine would be allowable (in the contract) only for a specific amount of time, such as thirty days, and the program would cause itself to self-destruct if it were used on a different machine for more than that time period.

Many of the technological techniques for limiting copying are being developed in the microcomputer area. Most microcomputer programs are distributed like books, without the economic leverage of providing updates or maintenance, and often are distributed so widely that enforcement is entirely impractical. For a brief time, popular microcomputer programs, such as

''Visicalc'' or ''Micro-Chess,'' were able to use the operating systems of certain computers, notably Apple II computers, to prevent themselves from being copied. Often this was done by manipulating the manner in which the program was stored on floppy disks. Recently, however, certain ''nibbling'' programs have been marketed which can defeat these methods. Much of the personal computer literature touches on this situation, pleading with the pirates to cease, since the protective mechanism they break means that the programs they soon make will also be stealable. Many of the pirates claim they break protective devices only to create back-up copies, and not to compete with the software houses. A new protective mechanism which has been proposed is to let the program allow itself to be copied several times before it implements the restrictive copying system.

More exotic self-help mechanisms can be contemplated. As computer systems become automatically connected to telephone and other communications networks, it would be possible to create a program which has the ability to dial its home office automatically and inform the home office that it has been stolen or is being used on the wrong machine. Another method is to scramble or encrypt the program code and to include the decoding chip when marketing the software to be inserted in the computer, or to use one of the ''double key'' systems in which the encoding key is well known but the decoding key is secret.[207]

Clearly, the best regulatory approach to these self-help mechanisms is not to specify what mechanisms are acceptable or to limit their use but to support their use.[208]

2. Trademark/Unfair Competition

Trademark protection can be important, particularly in the mass market for videogames and home computer programs. A substantial portion of sales may be lost if a version of a videogame which does not rise to the level of a copyright infringement is marketed under the same name, such as ''Pac-Man II.''[209] Similarly, use of unfair competition can restrict the passing off of one videogame or program for another. Also, certain elements of unfair competition, such as misappropriation of ideas or unjust enrichment, may in the appropriate context be arguable for protecting a program. For example, use of a program to create a new program may constitute unjust enrichment.[210]

[207]*E.g.*, Patent No. 4,168,396.

[208]*E.g.*,National Subscription Television v. S & H TV, 644 F.2d 820 (9th Cir. 1981) (interpreted Communications Act of 1934, 47 U.S.C. § 605, and concluded that the Act prevented the unauthorized sale of decoders which could unscramble pay TV signals.

[209]*E.g.*, Midway Mfg. Co. v. Bandai-America, Inc., 546 F. Supp. 125, 155-58 (D. N.J., Jul. 22, 1982) (summary judgment granted on infringement of trademark ''Galaxian'' but not on copyright infringement of ''Galaxian'' videogame).

[210]*E.g.*,Computer Print Sys., Inc. v. Lewis, 212 U.S.P.Q. 626 (Pa. Super. Ct. 1980) (company which received source code innocently subject to liability for unjust enrichment in developing competitive program). Note that unfair competitions laws may be preempted by § 301 of the 1976 Copyright Act. *E.g.*, Schuchart & Assoc. v. Solo Serve Corp., 540 F. Supp. 928, 942–48 (W.D. Tex. 1982) (unjust enrichment not preempted; misappropriation/unfair competition preempted).

3. International Protection

An important consideration is whether a program will be eligible for protection overseas.

a. *Patent*

It is unlikely that patent protection will be available, although interestingly enough, a patent was granted in Germany for the same program denied a patent in *Gottschalk v. Benson.*[211] Since then, the German patent law has been amended to exclude "programs for data processing installations."[212] In the European Economic Community (EEC), a new patent procedure allows a patent to be valid in all EEC countries with one filing, but excludes software "as such."[213] In France, a 1968 patent law excluded software.[214] This law was followed in Austria, Switzerland, Poland, the Netherlands and Denmark; on the continent only Sweden appears receptive to program patents.[215] The U.K., once receptive to program patents, is no longer.[216] In all these countries a U.S. patent can constitute full disclosure without protection.

b. *Copyright*

It is increasingly likely that software will be protected by copyright in the major Western developed nations. Recent decisions in France, Germany, Japan, and South Africa have held that software is copyrightable under their respective copyright laws.[217] Recent decisions in the U.K. indicate software for the most part is being protected under the U.K. 1956 Copyright Act even in the absence of a clear holding in favor of such protection.[218] These cases find soft-

[211]Western Elec. Co. Patent Appl. P-14-74-091.1-53 (Fed. Rep. Germany Patent Court, May 28, 1973), *as translated and reported in* FREED, COMPUTERS AND LAW 27 (5th Ed. 1976) (patent granted for same program denied a patent in *Benson*); *but see* Siemens A.G. v. Aeg Telefunken, FED. REP. GERMANY (S. Ct., June 22, 1976).

[212]1978 Patent Act § 1(2), No. 3.

[213]1973 Convention of Munich § 52(2)(c).

[214]French 1968 Patent Law, art. 7.

[215]Soltysinski, *Computer Programs and Patent Law: A Comparative Study*, 3 RUTG. J. COMPU. & L. 1 (1973).

[216]U.K. 1977 Patent Act; *Cp.* Burroughs Corp. (Perkins) Application, [1974] RPC 147, [1973] Fleet St. Rep. 439 (Pat. App. Trib. Jul. 30, 1973) (method of transmitting data to slave computers patentable).

[217]P. v. BMV, 46 Expertises les Systemes d'Information 243, 245 (Paris Ct. App., Dec. 1982) (dispute between applications program developer and a user); VisiCorp v. Basis Software GmbH, *reported in* 9 COMP. L. & TAX REP., No. 8, at 4, (March, 1983) (1st Mun. Dist. Ct., Dec. 21, 1982) ("VisiCalc" copyrightable as a "linguistic work of a literary nature" under § 2 of the German Copyright Act); Taito v. I.N.G. Enterprise reported in *Japan Times*, Dec. 8, 1982 (Tokyo Dist. Ct., Dec. 6, 1982) (microcomputer program embodying videogame copyrightable); Northern Office Micro Computers v. Rosenstein, 1981 (4) CPD 123, [1982] Fleet St. Rep. 124 (So. Africa 1981) (medical applications program source code in written and machine-readable form copyrightable under 1978 South African Copyright Act): *See generally* 9 COMP. L. & TAX REP., No. 8, at 4–5 (March 1983).

[218]*See* Systematics Ltd. v. London Computer Centre Ltd., available on LEXIS, *reported in*

ware copyrightable because the creative choices of a programmer are not narrowly channeled due to machine constraints but contain the stamp of individual style.

This is a welcome development, particularly because there are several international conventions which make the international protection of copyrighted works feasible. The United States is a member of the Universal Copyright Convention (UCC), of which most developed nations are members, and which requires only the use of a copyright notice, including the sign c-in-a-circle, in order to comply with all formalities of each local member country.[219] (Technically, "(c)" will not suffice.) The United States is also a member of the Buenos Aires Convention, whose only formality is to require a reservation clause such as "all rights reserved."[220] The United States is not a member of the Berne Convention, which requires no formalities for protection.[221] The 1976 Act contains certain changes, such as the duration of copyright, which would make it easier for the United States to join that Convention. In the meantime, it is suggested that to retain protection in Berne countries one publish simultaneously (i.e., within thirty days) in a Berne country (such as Canada) when first publishing anywhere in the world.[222]

These conventions will provide protection for software only if the country in which protection is desired recognizes protection. The recent decisions in major Western nations should create momentum leading to acceptance of copyright in most countries. Even then, certain peculiarities in the conventions may complicate the use of copyright across the borders. The UCC, for example, is in many respects structured like the 1909 Copyright Act in the United States, in that protection appears to attach only upon publication, and the notice provided under the UCC is not effective unless it is placed upon works when first published in visually perceptible form;[223] this would seem not to apply to unintelligible binary form which is marketed as an unpublished work. Still, since the notice is required only if the country in question requires formalities, and most developed nations other than the United States have none, this ambiguity in the UCC mostly affects works first published outside this country.

DataLink, 13 Dec. 1982 and in *The Times*, 16 Nov. 1982 (Ch. Div., Nov. 11, 1982) (CP/M program copyrightable; reported decision only discusses award of costs); Format Comm. Mfg. Ltd. v. ITT (U.K.) Ltd., available on LEXIS (Ct. App., Oct. 26, 1982) (telephone switching software; appeal of discovery order); Sega Ent. Ltd. v. Alca Elec. Ltd., [1981] Fleet St. Rep. 516 (Ch. Div., Mar. 31, 1982) ("Frogger" videogame stipulated copyrightable; decision involved enforcement of interim equitable relief); Gates v. Swift, [1982] RPC 339, [1981] Fleet St. Rep. 37 (Ch.Div., Oct. 27, 1980) (interim seizure of cassette tapes containing two Tandy TRS,VB42-80 operating systems programs and three "Adventure" videogames). *But see* Williams Elec. Inc. v. Rene Pierre S.A., available on LEXIS, MM No. 113 (Ch. Div., Apr. 29, 1981) ("Defender" videogame; dismissed for failure to establish jurisdiction over foreign defendant).

[219]Universal Copyright Convention, art. III(1).

[220]Buenos Aires Convention, 38 Stat. 1785 (1910).

[221]Berne Convention for the Protection of Literary and Artistic Works, art. 5(2) (Paris Act).

[222]*Id.*, art. 3(1)(b).

[223]Universal Copyright Convention, art. VI.

c. *Trade Secrets*

Besides copyright, a method for overseas protection of software is the use of trade or industrial secret laws. Such laws are present in most developed countries. The main impediment to the use of trade secret laws overseas is their weaker enforceability due to some of the differences between civil law and common law jurisdictions.

In civil law jurisdictions, it may be more difficult than in the United States to gain protective orders, which are essential in litigation; otherwise, the act of enjoining a trade secret infringer would result in the trade secret being part of the publicly available record of the lawsuit and would therefore constitute public disclosure and loss of future protection. Also, in many jurisdictions, discovery is not sanctioned or will not be as extensive as allowable in the United States, and it may be difficult to prove theft of trade secrets if the infringing products look very different. One informal practice is to appoint a special commissioner to analyze, in secret, the original and the alleged copy of the program. This both keeps the trade secret secret and can substitute for much of what would be required in discovery.

Trade secret enforcement can be enhanced through arbitration. Arbitration proceedings can be specified as being secret and discovery can be provided for. Other advantages include specifying that English shall be the language for all testimony and the language into which all documents shall be translated; that venue shall be in a specific place; that certain law shall be used, disregarding local custom and rules; and that the decisions and the reasons for the decision will be in writing (it is very disconcerting to go through a lengthy and complicated arbitration and receive a decision without any explanation or ruling as to facts or law). The arbitrator can also be specified, as, for example, a person having fluency in English, experience with data processing and being impartial in the arbitration (by international custom, even a party-selected arbitrator is supposed to act impartially unless the arbitration clause provides otherwise). Enforcement can also be easier for arbitration awards than for court judgments. Arbitration awards are in many countries treated as contractual matters, not judicial, and the sovereignty of local courts is not called into question. If the arbitration clause provides that the arbitration award can be enforceable in any competent court and is governed by the 1958 U.N. Convention, it is possible to enforce awards in member countries by going directly to the nonvenue country's court without first docketing the award in local courts.

d. *Registration Systems*

In many South American countries and other Third World nations, the transfer of technology from the developed world to the Third World is an important issue. Many of these countries have implemented registration systems under which any technology-related agreement must be scrutinized and approved by a local ministry.[224] These laws are intended to prevent the exploitation of local workers, for example by the creation of a factory, its use over many years, and the sudden withdrawal of the factory to a more favorable country, leaving in its wake semitrained workers and no ability to duplicate the secret processes. The registration process can be expensive and time-consuming. Many requirements to achieve registration can apply, some of which can lead to the loss of the technology at the termination of the relationship and others of which limit the relationship to a short time period, such as five years.[225] This may result in public disclosure of the trade secrets in the software and loss of trade secret protection both in that particular country and eventually around the world.

There are many suggestions on how to avoid this scenario, most of which are not entirely satisfactory. For example, the simplest method would be to use self-help: provide only object code of an old version of the software and not provide updates, so that by the end of the five-year period any loss of protection or disclosure of the old software would not be threatening to newly developed software. Another method is to negotiate special provisions with the registration authorities, which may require extralegal activity to implement. One idea which to the knowledge of this author has not been attempted is to "leapfrog" the software license by providing that within each five-year license a new license will be executed for an updated version of the software and providing that the old version shall be returned in its entirety upon such exchange. Thus, a five-year license can become a perpetual license, with continued legal protection.

Even if a protective method were developed, there can be limitations on enforcement. There may be a limited right or no right to injunctive or other similar relief. Damages may not be adequate compensation, particularly if trade secrets are compromised worldwide. In some countries the best interim remedy may be what is in some jurisdictions called *secuestro,* the ability to seize all materials embodying the subject of the license in the event of any breach. This too has its problems, including the practical problem of ensuring that all machine-readable copies have been deleted from the offending party's computer system. *Secuestro* perhaps is best used to stop the wholesale competitive marketing of the software.

e. *International Trade Commission and Customs Service*

The International Trade Commission can prevent the import of products involving unfair methods of competition.[226] This can include unlicensed import of a product in violation of a patent,[227] or following misappropriation of a trade

[224]*See, e.g.*, Venezuelan Decree No. 2.442, art. 63 *et seq.* (Nov. 15, 1977).
[225]*Id.*, art. 65(e).
[226]19 U.S.C. § 133.
[227]*In re* Chain Door Locks, 191 U.S.P.Q. 272 (ITC 1976).

secret.[228] It may also include importation of a copyright-infringing product, although typically the quicker and less expensive way to prevent such import is to use the Customs Service.[229]

PROPOSED REGULATORY SCHEMES

1. Proposals

a. *ADAPSO Proposal*

The Association of Data Processing Service Organization (ADAPSO) has investigated its members' problems relating to the protection of software and recently proposed five basic amendments to the 1976 Copyright Act:

1. to specify that copyright notice does not imply public disclosure or publication;
2. to cover all forms of computer programs, including program descriptions other than source code;
3. to specify that trade secret law is not preempted;
4. to allow the symbol (c) in addition to the symbol ©; and
5. to allow registration of programs in secret.

These proposals were incorporated into H.R. 6983, introduced on August 12, 1982. This bill was not acted upon, and ADAPSO has prepared a different bill which it may submit to the current Congress. This new bill corrects some of the problems in ADAPSO's Proposal. ADAPSO has amended its Proposal to support the changes.

Although each of the areas of ADAPSO's Proposal would be quite useful in clarifying certain problems with respect to the coverage of copyright law, ADAPSO's recommended language is not always well considered. Regarding preemption, instead of simply stating that "trade secret law" is not preempted by the 1976 Act, ADAPSO recommended that there would be no preemption with respect to "trade secret law not equivalent to copyright." This definition unfortunately simply repeats one of the unsolved issues in the 1976 Act: whether trade secret law encompasses rights that are not equivalent to the rights granted by copyright law. If it is desired that both systems of protection coexist, it would have been better for ADAPSO simply to have stated that trade secret law is not preempted, and to allow the much less troublesome problem of whether certain state laws are "trade secret" laws or are what used to be called common law copyright. Since trade secret law is well defined in the United States, and historically was easy to distinguish from common law copyright,

[228]*In re* Apparatus for the Continuous Production of Copper Rod, 206 U.S.P.Q. 138 (ITC 1979).

[229]19 C.F.R. § 133.1 *et seq. See In re* Coin-Operated Audio-Visual Games and Components Thereof, Investigation No. 337-TA-87, USITC Pub. L. No. 1160 (June 1981) (Galaxian); *In re* Certain Coin-Operated Audio-Visual Games and Components Thereof, Investigation No. 337-TA-105, USITC Pub. L. No. 1267 (July 1982) (Rally-X; Pac-Man).

this dispute would not cause the same problems to software houses as are caused by the current law or would be caused by ADAPSO's recommendation.

The revised bill does not clearly remove this problem. The new language stated that Section 301 will not "limit any right or remedy which the owner of a copyright may have under state trade secret or other law not equivalent to any of the exclusive rights within the general scope of Copyright. . . ." It is ambiguous whether "not equivalent to" applies to "trade secret . . . law" or just to "other law." The revised ADAPSO Proposal indicates that it does intend to preempt trade secret rights equivalent to copyrights. ADAPSO believes this will cause software trade secret agreements to be construed so as not to conflict with copyright. This desire is laudable, and if copyright is interpreted broadly enough to protect the specific logic and design of a program rather than just the code, it is feasible that both protective systems could complement each other. Nevertheless, this new language will still allow for fine distinctions between trade secrets and copyrights, which will continue the uncertainty as to the extent of preemption and may cause lack of protection in some cases.

Regarding the attempt to cover all forms of computer programs, ADAPSO recommended first that new definitions be included of *computer program, program description,* and *supporting materials. Computer software* was then defined to be the "literary works in" the other three definitions. This simply begs the question as to whether computer software in each of its forms is a literary work. (The WIPO Proposals, discussed below, from which these four definitions were derived, defines *software* to mean any or several of the "items" in the other three definitions.) In a companion recommendation, ADAPSO would add to the enumerated works of authorship in Section 102 of the 1976 Act, in lieu of the phrase "literary works, including computer software." Because of the tautological definition of computer software, however, expressly adding "computer software" as a type of literary work adds nothing.

The revised bill obviates these problems by removing the proposed amendment to Section 102 and by replacing the four definitions of types of software with a new definition of *computer program.* Still, because it is still disputed whether all types of software contain "authorship," it might have been better for ADAPSO to have added as an additional type of work of authorship a category called "computer programs." Also, the new definition of *computer program* may be a step backwards. The current definition is functional; in effect, a program is defined as a set of instructions which could operate a computer. The proposed definition is partially formal, restricting the current definition to programs in a "verbal, schematic or other form." This confuses the *work* with the *embodiment.* ADAPSO's intent was to combine "computer program" and "program description," to indicate that the protected program could exist in computer-readable form as well as human-readable and schematic form, such as a flow chart. If "program" is understood to be the specific logic and design of software, then this combination of definitions is unnecessary. A flow chart with sufficient detail could express the same logic and design as a machine-readable program and be protectable as a "program" regardless of ADAPSO's new definition.

The addition of "(c)" as a copyright symbol would be welcome, although hardly necessary. ADAPSO should consider whether it should also propose allowing "(P)" for the performance copyright symbol Ⓟ and "(R)" for the trademark symbol ®.

The ADAPSO Proposal calls for the Copyright Office to create a nonpublic registration system, a welcome change. However, the enabling amendment may not be a specific enough clause to avoid potential Freedom of Information Act problems, discussed above.[230] ADAPSO should also reconsider whether to recommend deposit of software, even securely, or to establish a system in which the registered copies are returned, not deposited.

Neither the ADAPSO Proposal nor the revised bill address several other unresolved copyright problems, including whether object code is a "copy" or "derivative work" of source code, what is the effect of the Copyright Office applying a "rule of doubt" to registration of object code, how to interpret Section 117(1) with respect to per-computer restrictions or ability to make adaptations, and how much revision to a program is necessary before it should be reregistered.

b. *WIPO Proposal*

The World Intellectual Property Organization (WIPO) coterminously with CONTU investigated the most suitable protection method for software on an international scale. Its 1977 *Report* contains a detailed and lengthy discussion of whether a patent-oriented or a copyright-oriented protective scheme would be most suitable for protection of software. WIPO concluded that a copyright-oriented scheme would be most suitable. Such a scheme would protect the expression of software, but not the fundamental ideas. WIPO also added to its copyright scheme certain trade secret concepts to enable the functionality of a program ascertainable from source materials and user documentation to remain protectable.

The WIPO proposals have interesting differences from the 1976 Copyright Act. No registration or deposit is required. This is typical of most foreign copyright systems. The *WIPO Report* questions administration of a registration system—whether programs could be filed and classified such that the advantages of registration would be available in reality. It also points out the administrative problem of the continual updating and modification of software. Unlike books, records and the vast bulk of copyrighted material, to register all versions of programs would cause a continual storage of minor modifications. It is difficult to draw a line between when an update should be deposited and when it should not. The *WIPO Report* also questions whether an optional deposit system—perhaps with the idea of disclosure of concepts or an abstract of the program as

[230]*See* Part 2 under "Trade Secrets."

identifying materials—would be useful. If the disclosure were sufficient to allow a competitor to recreate the same program, it is unlikely to be deposited.

WIPO would cover software in its broadest sense, and include programs, program descriptions and supporting materials.

The WIPO proposals would clarify the scope of coverage and may avoid a generation of litigation. Besides preventing copying, it adds "trade secret" concepts: it prevents disclosure of a copy either orally or physically and prevents using the program description to create a "substantially similar" program. It also expressly covers translations by use of computer programs which compile, assemble, decompile, disassemble or cross-compile from one language to another. It covers the use of the program in a computer and would continue current marketing practices of restricting licenses to a per-computer basis.

The coverage of protection under the WIPO proposals is twenty years from use or twenty-five years from creation. This is longer than patents (seventeen years)[231] but shorter than copyright (seventy-five years if a work for hire, otherwise life of author plus fifty years).[232] The stated reason is that most programs are of an industrial nature and have a limited product use lifetime.

Finally, the WIPO Proposal would provide no statutory damages or attorneys' fees to the prevailing party, and would make injunctive relief discretionary to the extent issuance of an injunction would be found to be "unreasonable."

The WIPO Proposal contains at least three questionable points. The inability to gain attorneys' fees, statutory damages and at times an injunction could put software owners at a disadvantage. The cost of litigation often exceeds actual damages, particularly if an injunction is sought at an early stage in a violation. Also, lack of injunctive relief in "unreasonable" cases may cause the software owner gradually to lose control over the software, making enforcement of rights, particularly trade secret rights, more difficult.

Second, the definition of *program description* would have to be understood narrowly. WIPO precludes using program description to create programs. If this right extends to input and output formats, programming techniques and other facets of software beyond its specific logic and design, too much protection may have been granted. Apple Computer, for example, might then be able to prevent others from developing Apple-compatible computers even if they rely solely on Apple's user documentation and the actual operation of an Apple computer. WIPO did not intend this result; the user manuals were to be considered "supporting materials" not "program descriptions." But in attempting to cover all aspects of protecting software, WIPO may have allowed this type of argument.

The third questionable aspect of WIPO's proposals is the lack of preemp-

[231]35 U.S.C. § 154.
[232]17 U.S.C. § 302.

tion. Patent, copyright, and trade secret protection can coexist with WIPO's *sui generis* scheme. This may complicate protection, rather than simplify it; software vendors in their marketing procedures, software licenses and litigation will attempt to enforce WIPO, copyright and trade secret protection together. The Achilles' heel in any *sui generis* system is lack of experience with the new scheme and the corresponding inability to foresee all problems. The resulting fear that some problem may have been overlooked leads to a desire to cover all bets by not preempting other protective schemes, perhaps creating a worse situation than that which now exists.

Therefore, the WIPO proposals would work best from a construction and interpretation point of view if they were appended to the copyright laws of member countries. In many respects, the ADAPSO proposals purports to create the same type of coverage specified in the *WIPO Report*. Specifically, both purport to clarify what it means to "copy" a program and to what extent ideas are covered rather than expression. Both also purport to allow the continuance of trade secret law concepts. Indeed, it might be possible for the United States to become a party to the WIPO Proposal by using the ADAPSO amendments, slightly modified, and the lead of the United States in this area might encourage other developed nations to follow to protect software.

c. *Computer Software Protection Act*

Commissioner Hersey of CONTU had the Computer Software Protection Act prepared as an alternative to the CONTU proposed amendments. The Act is a copyright approach which does not have any examination of programs as in a patent system. Instead, there would be a registration system in a special registry in the Department of Commerce.

The Act is different from copyright in a number of respects. Protection would only be for ten years. All other protective schemes would be preempted, except trademark protection and copyrights in the descriptive parts of documentation. A special notice would be created, like a copyright notice, involving the symbol s-in-a-circle. The Act would cover both the program's code itself and "the original method or process embodied in the software" other than embodiments of mathematical relationships or scientific principles.

Unfortunately, this Act suffers from many of the same deficiencies that exist under the current system, including lack of clarity in coverage. In any event, the Act has dropped from sight and consideration and was not even appended to the *CONTU Report.*

d. *IBM Proposal*

IBM Corporation proposed a patent approach to software protection over a decade ago. Its key features were requiring deposit of the program and publication of key, innovative concepts in an abstract. It was essentially a proposal for nonprotection, since it provided broad grounds to invalidate registration or to defend against an infringement action. Also, the disclosure of the concepts created the ability of independent creation of competing programs, a situation which would aid hardware vendors (because the proliferation of programs sells hardware), not software houses.

This proposal was offered at a time when IBM and the other hardware vendors were not inclined to allow the protection of most software, in order to sell their hardware. Not surprisingly, this proposal can be criticized for being helpful only to large companies that have the resources to detect and enforce the rights granted. Cottage industry software houses would have to rely on secrecy instead since they would have insufficient resources and information to detect or enjoin infringers, and disclosure of the key ideas would enable many competitors to enter their markets.

This proposal has died a natural death and is unlikely to be considered in any form in the future, nor to be proposed by IBM, which now has different interests regarding the protection of software.

2. Implementing the Reforms

The most direct manner of implementing any of these or other reforms is by legislation in Washington. Because the pending ADAPSO Proposal is likely to be submitted as a bill in Congress, reconsideration of this method of implementation is timely. If any deficiencies in the new ADAPSO Proposal are not cured but pass into law, the goodwill of the software industry in asking for further revisions will probably have been exhausted.

The problem with legislation is best exemplified by the ADAPSO Proposal itself. The ADAPSO amendments appear to represent a compromise of various positions, and some of the ambiguities in ADAPSO's proposed amendments are presumably the result. The hearings on the new copyright legislation will likely have conflicting comments filed by the Department of Justice, by large vendors, by the Copyright Office, and by a myriad of others. Consequently, it is possible that if additional points are raised for legislation, the same types of compromises will be forced upon the new recommendations. Also, it is possible that there would be a change, perhaps on the floor of the houses in Congress, which has not been well considered. For example, the change from ''rightful possessor'' to ''owner'' in the 1980 copyright amendments was made at the last minute without substantial explanation in the record. Similarly, when the 1976 Copyright Act was being considered, the preemption section, Section 301, was changed at the last minute. In other words, a worse situation may be created than now exists.

The problems in this area perhaps can best be solved on a case-by-case basis, in which the functional impact of a particular change is clear and limited to particular circumstances. A sweeping theoretical change does not have this functional approach and does not have the fundamental appreciation of the effects of the amendments. This is not an avoidance of the political process; it is a practical recommendation based upon the conflicting and compelling interests in this area and the difficulty in appreciating the nature of software.

Therefore, ADAPSO's bill should be limited to items requiring legislative action, such as a secure deposit system. The interpretive questions would be left for common law development.

3. Conclusion

The main problem addressed by these proposed protective schemes is the fundamental issue of what is being protected: the instructions themselves (the "expression" of the program) or the functionality of the program. From a pro-competitive point of view, since the software industry has burgeoned without having any patent-like protection or the protection of functionality, it is questionable whether such protection is necessary. The critical concern of the industry is copying or direct reverse engineering of software, not the learning of certain ideas and functionalities and attempting to recreate them. Access to software or to materials describing software is a critical first step to copying or reverse engineering. Therefore, of all these systems, copyright law as interpreted above or the WIPO Proposal as improved above are probably closest to what is desired by the industry.

As a consequence, it is recommended that the WIPO Proposal as improved above be seriously considered by Congress, be implemented as part of the 1976 Copyright Act if possible, and be pursued worldwide. As a compromise, the ADAPSO Proposal could be improved, with Congress concentrating on the following points.

First, it is important to allow copyright registration with secrecy. This can best be accomplished by requiring no registration at all or by requiring only an inspection by the registration office and return of the software. The purpose of the secrecy is both to allow the further enforcement of trade secret rights and to avoid the problem faced by many small software houses of not having adequate resources to detect and enforce their rights.

Second, it will be quite valuable to implement any protection scheme on a worldwide basis. The easiest manner to do this may be to implement WIPO's proposals through existing copyright laws, thus having an existing body of law from which to plan and having the ability to use worldwide copyright conventions as well as WIPO treaties to implement protection. Alternatively, current international copyright conventions should be clarified to allow the overseas protection of software. In this regard, it would be helpful to amend the Universal Copyright Convention to allow for the use of the symbol "(c)" in lieu of the symbol "©" and to create the concept of placing the copyright notice on unpublished works in machine-readable form.

Third, trade secret law should not be preempted in whole, nor should use of copyright be considered an election or give rise to an estoppel, unless trade secret concepts which support current licensing practices are added to the protective scheme, as in WIPO's proposals. In particular, a copyright violation should occur if a competing program is made from source or object code or specific and complete program descriptions. However, if the copy was made only from supporting materials or by appropriating the functionality of the software and not the particular way in which it was expressed in the design of the program, no copyright infringement would occur. These problems under current law can best be left to common law development; amendment to the Act would only be needed to correct mistaken precedent.

Fourth, those types of software which are marketed with restrictive licenses should be deemed not to be "published." This would include a broad reading of the "limited publication" rule and would include that use of a copyright notice does not constitute an admission of publication or public disclosure which would compromise the software's status as an unpublished work or as a trade secret. These problems are being satisfactorily answered by recent cases.

Finally, additional questions, such as what constitutes a "substantially similar" copy or what is the legal relation between source and object code, can best be answered on a case-by-case approach.

With these changes, judicial or legislative, a relatively satisfactory scheme of protection would be created. Patent law would apply not to software *per se* but to specific machines or processes in which software is one element that operates on a real-time basis affecting the other elements. Copyright law would protect all materials from which symbolic forms of programs are reproducible. Trade secret law would protect input and output formats, underlying algorithms, concepts and techniques, and source materials, and would restrict disloyal employees. International law in the developed countries would be in accord with these copyright and trade secret interpretations. Finally, in light of the practical limits on legal protection, technological methods of protection would be available. The peaceful coexistence of these legal doctrines should allow the software industry to prosper and software users to benefit.