Volume 5, Fall Issue, 1991

COPYRIGHT OR PATENT OR BOTH: AN ALGORITHMIC APPROACH TO COMPUTER SOFTWARE PROTECTION

John Swinson *

INTRODUCTION

The creation of an efficient legal protection system for computer programs has been a difficult process. The growth of programming occurred rapidly. Lawyers knew little about computers and computer scientists knew scant about the details of the law. Because of this lack of knowledge, lawyers pigeonholed computer programs into the existing intellectual property framework of copyright and patent without acknowledging the potential problems. Legal rules relating to other types of works such as novels and plays were applied with little adaption to computer software. The chaotic expansion of protection caused by the transfer of these preexisting rules to the computer programming field continues to worry many computer scientists.

The theme of this Article is that algorithms, fundamental to the growth of computer science, must be understood and taken into account to sensibly formulate and apply a legal protection system to computer software. An algorithm can be expressed in different ways or at different levels of abstraction. When asked to detail a solution to the problem, that solution will be written at various levels of sophistication by people using divergent expressions, but an intelligent human should be able to examine the solutions and determine which are the same. Can the law do likewise for computer programs and algorithms?

The Article starts with a brief introduction to algorithms and computer programs. It then analyzes whether the application of the patent system to algorithms and computer programs provides a coherent, reasonable approach to the legal protection of computer software. The software industry's criticisms of the application of the patent system to computer software, namely the inapplicability of traditional patent justifications to this field and the current patent system's counter-

---

* Barrister-at-law, Queensland, Australia; B.A. Computer Science, 1986, University of Queensland; LL.B. (Hons.) University of Queensland, 1988; LL.M., 1991, Harvard Law School. The writing of this Article was made possible through the support of the Fulbright Foundation, the Frank Knox Memorial Fellowship, and Mallesons Stephen Jaques. The author wishes to thank Professor Terry Fisher of the Harvard Law School, and Mr. Peter Treyde of the Attorney General's Department of Australia for supplying materials concerning the Copyright Law Review Committee.

productive effects on the growth of this industry, are shown to be little different than grievances aimed at the patent system in general. Yet, the unique nature of computer programming exacerbates traditional faults in the system.

Before resolving the issue of whether patents should be granted for computer programs and algorithms, the other major form of protection, copyright, is examined. The Article assumes the desirability of some protection for software to encourage innovation. The contentious issue becomes the extent of protection. What should be protected as a computer program? How can one tell if two programs are the same? Should the test be whether they perform the same function or whether a user believes them to be the same? An examination of the law in the United States and Australia, where the copyright systems are somewhat similar but where different problems have stretched them in different directions, reveals the need for one set of rules that can be applied in both jurisdictions. The policy arguments for and against protecting user interfaces will be outlined. An algorithmic approach will be used to formulate the optimal protection plan to be applied to computer programs.

In conclusion, the Article suggests that only a modified version of copyright protection is needed for the proper level of protection of computer software. The value of software is its form of expression—a computer can understand and carry out instructions expressed as a program. The thesis of this Article is that patent protection is too broad and that copyright, if limited, will provide the necessary incentives in an efficient way to encourage progress in software development.

I. BACKGROUND

A. Algorithms

In order that legal rules can sensibly be applied to computer software, a system of intellectual property law, and the lawyers practicing it, must comprehend and account for algorithms. An algorithm is simply a series of steps telling a processor how to perform a given task. For example, the algorithm for knitting a sweater is a knitting pattern, in which a typical step is "knit one, purl one," and the processor is a human being. Recipes, instructions to build model planes, and computer programs are all algorithms.

An algorithm can be expressed in different ways, using divergent expressions at varying levels of sophistication and abstraction. For example, an algorithm for sorting a list into order may be expressed in the form of a computer program, in which case a typical step may be

"while n > 0 do," and the processor will be a computer. That same algorithm may be written for a human to carry out, but the language used would be different, enabling the human processor to understand the algorithm. Still, the two expressions would be the same algorithm.

The steps of the algorithm must be written to be understood and executed by a processor. Algorithms are not dependent on digital computers. Many algorithms were written for humans, long before computers were invented, by people such as Pythagoras, Beethoven, and Newton. Algorithms are processor-independent in the sense that the same algorithm can be carried out by many different types of processors, provided that the processor understands the form of expression used to communicate the algorithm.[1] Each algorithm must be designed to tell a certain processor what to do, but the processor need not be specified. However, as instructions for accomplishing a task, all algorithms by definition have at least one defined or implied processor.

A processor must be defined for an algorithm to exist, but how specific does the definition of the processor have to be? Some argue that a series of instructions, written in a high-level form of expression, that can be carried out by a "generic" machine, is not an algorithm, but rather a law of nature, a mathematical formula, or an idea. Because an algorithm must be tied to a specific device that will be used in the problem-solving process, "until the device is specified, an algorithm cannot be constructed."[2] If this is true, is a program written in a general high-level language that can be executed by any computer with the correct compiler just an idea? High-level languages are designed so that they are not device-specific. The better view is that at least one processor must be defined or implied for an algorithm to exist. In identifying a processor, the higher the level of abstraction of the algorithm, the lower the level of specificity needed in defining the processor.

Every computer program is an algorithm.[3] For computers to perform any useful task, they need to be instructed what to do. These instructions

---

1. The idea of the algorithm, or the general description of the solution to the problem can be expressed in different languages for different processors, but the algorithm is still the same. For example, a recipe (which is an algorithm) to bake a cake can be written in English or French, and so long as the English cook and the French cook can understand the language, the same cake should result. In fact, given any algorithm, it is possible to code it in any programming language. *See* Allen Newell, *Response: The Models Are Broken, The Models Are Broken!*, 47 U. PITT. L. REV. 1023, 1029 (1986).

2. Mitchell P. Novick & Helene Wallenstein, *Algorithm and Software Patentability*, 7 RUTGERS COMPUTER & TECH. L.J. 313, 335 (1980).

3. *See* Newell, *supra* note 1: "An algorithm is just an abstract program . . . the only distinction is the degree of abstraction." *But see* Paine, Webber, Jackson, & Curtis, Inc. v. Merrill, Lynch, Pierce, Fenner, & Smith, Inc., 564 F. Supp. 1358 (D. Del. 1983) (Defining "algorithm" narrowly, the court held that a computer program is not an algorithm.).

make up the algorithm called the program. However, algorithms for which a computer is the intended processor are no different from other algorithms, except for the manner and level of expression.[4] A computer program may just be one form of expression of an algorithm.

The travelling salesman problem offers an illustration.[5] The solution, an algorithm, is expressed by network theorists partly in English and partly in symbols. The algorithm could be used by the post office in an instruction manual for letter carriers telling them how to determine the most efficient delivery route: That use would be one manner of expressing the algorithm. A computer scientist may take the algorithm and, with only a "generic computer" in mind as the intended processor, express the algorithm in pseudo code (an abbreviated form of English) or a flow chart. At this stage, the algorithm is expressed in a form of code at a high level with no particular processor in mind. After the intended application and the programming language are chosen, the algorithm is refined, step by step, until it is in a form understandable to the computer (the computer program). The level of expression depends on the level of sophistication of the chosen computer language.[6] The identical algorithm could be written in a second programming language. The expression would be different, but the fundamental idea (of how to solve the problem) and the results of running the program would be the same. There is nothing to stop the programmer from refining the algorithm still further, so that it is expressed in a less advanced computer language, such as machine code.[7] The refinements express the same algorithm in a more detailed way, or as described by computer scientists, at a lower level. There is a continuum between the high-level description of the solution to the problem and the low-level machine code. The only change is the detail of expression.

---

4. It was predicted in 1980 that if "a court determines that a program is identical to the algorithm it expresses, then the court will find the program unpatentable." Michael C. Gemignani, *Legal Protection for Software*, 7 RUTGERS COMPUTER & TECH. L.J. 269, 294 (1980). Such a test would render all programs unpatentable; a program is a way of expressing an algorithm so that a computer can understand it.

5. The algorithm determines the shortest route around a network so that the "salesman" visits all "towns."

6. As technology advances, the detail of expression required to communicate with and program a computer will decrease. High-level languages look more like natural languages than like machine code. Some database query languages allow questions to be asked in natural languages such as English. Any dividing line between an algorithm written in a language that only a computer can understand and one that a human can understand is disappearing rapidly. Indeed, this is a goal of computer science.

7. This is, in effect, what a program called the compiler does.

B. Computer Programs

A computer program uses a number of algorithms to produce a certain result. This fact prompts some commentators to advocate protection of the function of the program, rather than its expression. However, since a given result can generally be reached by more than one program, several complications are created for such a function-based definition. For example, a second programmer may write a different program that, to a user, operates in the same way and has the same user interface.[8] The program, although achieving the same end, may do so by a completely different route or, in other words, by using a different algorithm. Alternatively, the algorithm may be the same, but the actual code may be different, because the programming language or operating system used is different, or because the programmers have different programming styles. Another variation occurs when the same algorithm is used to accomplish the same result but the user interface and output are different; the user would then be unaware that the algorithm is the same. Finally, two programmers may use different algorithms and interfaces, but write programs that accomplish identical goals. For example, both programs may produce useful airline boarding passes with the same information, but the programs look different to the user and result from dissimilar algorithms.

To say that one computer program has the same function as another program really says nothing about the expression used in the program, the expression produced by the computer as output, or the algorithm used when coding the program. Simply, the function of a program is its purpose, as distinguished from how it accomplishes that purpose (the algorithm) or what is produced (the output).

It is often stated that the underlying algorithm of a computer program is the idea, and that the computer program is the expression of that idea. This is an over-simplification. A program may contain many algorithms, to control the data flow, to control the screen display, to sort things into order when needed, and to accomplish the overall task. Some of a program's algorithms may perform a very small part of the overall function, whereas others may define the whole operation of the program. All of these individual algorithms can be expressed at different levels of abstraction. The computer program as written is only one of the possible expressions. In one sense, the computer program is the only expression

8. The user interface is the program's external appearance; two programs with identical interfaces would appear, to the user, to be identical, even though they might have completely different internal program workings.

that correctly maps the algorithm of the program. It is convenient, however, to think of the unexpressed method of accomplishing the task as an idea; to call the algorithm of the program an idea merely states a conclusion and tells nothing about the idea itself.

In summary, one should understand the following basic features of algorithms:

a. An algorithm is a set of instructions that are followed by a processor to carry out a process, which need not have anything to do with mathematics.
b. Algorithms are not dependent on having a digital computer as the processor.
c. Algorithms are fundamental to computer science. Every computer program is the expression of at least one algorithm.
d. Algorithms can be used to solve many problems, not just mathematical problems.

II. PATENT

Computer software, like any other invention, is currently the proper subject for patent protection if it is a "new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof."[9] Excluded from patent protection are laws of nature, natural phenomena, and abstract ideas.[10] There has been much debate as to whether algorithms and computer programs are more like processes and machines, therefore eligible for patenting, or more like the laws of nature, therefore unpatentable.[11] Part of the confusion has been caused

9. Patent Act § 101, 35 U.S.C. § 101 (1988).

10. *See* Le Roy v. Tatham, 55 U.S. (14 How.) 156, 175 (1852) ("A principle in the abstract . . . cannot be patented."); Diamond v. Diehr, 450 U.S. 175 (1981); Parker v. Flook, 437 U.S. 584 (1978).

11. *See, e.g.*, Gregory J. Maier, *Software Protection—Integrating Patent Copyright and Trade Secret Law*, 69 J. PAT. & TRADEMARK OFF. SOC'Y. 151, 165 (1987) ("patent protection is presently available for virtually all software inventions"); Alan C. Rose, *Protection of Intellectual Property Rights in Computers and Computer Programs*, 9 PEPP. L. REV. 547, 556 (1982) ("at least some subject matter involving computers may be patented"); Jack E. Brown, *The Current Status of Copyright and Patent Protection for Computer Software*, 12 COMPUTER L. REP. 406, 407 (1990) ("Provided it is not expressed as a pure mathematical algorithm, software that qualifies as nonobvious invention also is protected by patent"); David Bender, *The Case for Software Patents*, 6 COMPUTER LAW. 2 (1989) ("'software patents' are often available on a cost effective basis and may be quite valuable"); Donald S. Chisum, *The Patentability of Algorithms*, 47 U. PITT. L. REV. 959, 960 (1986) ("mathematical algorithms 'as such' or 'in the abstract' do not constitute patentable subject matter"); Comment, *The Patenting of MIS Computer Programs*, 21 PAC. L. J. 761, 762 (1990) ("no court has been willing to grant patent protection to a computer program of and in itself").

by the judicial system's unfamiliarity with algorithms.

A. Algorithms and Patent Law

Courts have problems with the term "algorithm." It is not defined in the Patent Act, nor has the Supreme Court considered the word in great depth. The Supreme Court, in its most prominent case on this question, adopted the view that an algorithm, being a "procedure for solving a given type of mathematical problem," is not patentable, but the application of an algorithm "t a known structure or process may well be deserving patent protection."[12] The Court is underinclusive in saying that an algorithm is a procedure for solving a mathematical problem, unless such procedures as knitting a sweater or building a model plane are regarded as mathematical problems.

The Patent Act does not explicitly prevent the patenting of algorithms. However, in practice, because of the lack of understanding of the distinction between algorithms and computer programs, and because the inherent nature of an algorithm is to carry out a process (which is one subject matter of patent), the distinction that the Supreme Court articulated has proven to be of little use.[13] The United States Patent and Trademark Office ("the PTO") has interpreted the Supreme Court's decision as allowing patents for computer software but has disregarded the limitations that decision imposes.[14] Many of the patents granted to date

---

12. *Diehr*, 450 U.S. at 187.

13. The Supreme Court decided that a claim "does not become nonstatutory simply because it uses a mathematical formula, computer program or digital device." *Id.* The Court held that insignificant post-solution activity will not transform an unpatentable principle into a patentable process, but when a claim containing a mathematical formula implements or applies that formula in a process that performs a function the patent laws were designed to protect (such as transforming an article to a different state or thing), then the claim satisfied the requirements of the Patent Act. The problem with the decision is that all computer programs are applied processes. The test has not been limited, in its application, to processes physically transforming matter, and was regarded as the "go-ahead" for patentability of algorithms and software. *See, e.g., In re* Pardo, 684 F.2d 912 (C.C.P.A. 1982); *In re* Abele, 684 F.2d 902 (C.C.P.A. 1982); *cf. In re* Bradley, 600 F.2d 807 (C.C.P.A. 1979), *summarily aff'd*, 450 U.S. 381 (1981) (no algorithm in an invention in firmware module that directs data flow transfers between register and memory); Paine, Webber, Jackson, & Curtis, Inc. v. Merrill Lynch, Pierce, Fenner, & Smith, Inc., 564 F. Supp. 1358 (D. Del. 1983) (suggesting that any new computer program capable of commercial use will be patentable, provided only that it avoids reciting a mathematical algorithm that was defined in a very narrow way). *See generally* COLIN TAPPER, COMPUTER LAW 20–22 (4th ed. 1989).

14. Many of the patents granted by the Patent Office "are 'pure' software patents which indicates the Patent Office is now willing to grant patents for novel and nonobvious computer programs operating on conventional off-the-shelf computer hardware." PROPRIETARY RIGHTS COMMITTEE, COMPUTER LAW SECTION, STATE BAR OF MICHIGAN, A SURVEY OF US SOFTWARE PATENTS ISSUED FROM JULY 1987 THROUGH DECEMBER 1987, *quoted in* Bender, *supra* note 11, at 4. *See also* U.S. PAT. &

are regarded by many computer scientists as patents for pure algorithms.

The PTO allows the patenting of algorithms, but not mathematical formulas. It regularly applies a two-step test to determine whether an invention involving a computer program is directed to statutory subject matter. The first step is to decide if the claims in the patent directly or indirectly recite a mathematical algorithm. For example, if the claim contains words or equations that look like a mathematical formula, the claim recites a mathematical algorithm.

Secondly, the claim as a whole is analyzed to determine whether it preempts the "algorithm."[15] The claims are looked at without the "algorithm" to see if what remains is otherwise statutory. If what remains is data gathering or non-essential post-solution activity, such as the transmission of data or the display of output, the claim is held to be non-statutory.

It would seem then that when the PTO talks of mathematical algorithms, it really means mathematical formulas. A recent decision of the Board of Patent Appeals and Interferences, *Ex parte Logan*,[16] has said that this is not so. The Board noted that mathematical algorithms could be computational procedures.[17] But the Board then held that the claims before it

> did not recite a mathematical algorithm, because neither claim essentially recites, either directly or indirectly, a method of calculation, i.e., a method of computing one or more numbers from a different set of numbers by performing a series of mathematical computations.[18]

This definition of a mathematical algorithm seems close to that of a mathematical formula.

The line the PTO draws is between algorithms and mathematical formulas. In effect, all algorithms, so long as not simply algorithms inserting data into a mathematical formula (or a computational procedure where the input and output are numbers) are patentable subject matter. The *Logan* test would render only a small number of claims non-statutory. Any claim where either the input or result of the process is not

---

TRADEMARK OFF., THE MANUAL OF PATENT EXAMINING PROCEDURES § 2106 (stating that the Patent Office readily accepts claims relating to programs as patentable).

15. *See In re* Iwashashi, 888 F.2d 1370, 1375 (Fed. Cir. 1989); *In re* Grams, 888 F.2d 835, 837–38 (Fed. Cir. 1989).

16. *Ex parte* Logan, Appeal No. 89–2047 (B.P.A.I. Feb. 20, 1991).

17. *See id.* at 6.

18. *Id.* at 10.

a number or a set of numbers would not recite what the Board calls a mathematical algorithm and is therefore patentable. That the algorithm has a computer as the processor "is not a proper basis for [a section 101] rejection."[19] An algorithm that is something more than the application of a mathematical formula, and that is capable of being expressed in the form of a computer program, is patentable.

Patent law also requires that the process being patented be useful.[20] Therefore, to be the proper subject of patent protection as a process, the patent application detailing the algorithm (or computer program) needs to specify, among other things, the processor to be used in the process. A processor must be defined for an algorithm to exist, but how specific does the description of the processor have to be? It could be argued either that an algorithm written at a high level of abstraction is patentable, since it describes how a problem could be solved or that it is so abstract that it is a non-useful or unpatentable idea.[21] The same algorithm may be able to be expressed so that processors other than computers can complete the process. The consequence of granting a patent monopoly over the use of such an algorithm would be the total restriction of that task, regardless of the processor contemplated for use, even when the processor is a human carrying out the process without the use of a machine. If patent law is to give coverage to algorithms and computer programs, it must demand that the processor be defined in detail and that the scope of protection be limited to the use of that algorithm on the specified processor. Anything else would risk overbroad protection.

B. Can Existing Patent Rules Deal with Algorithms?

One goal of patent law is to encourage the implementation of knowledge for the creation of useful products, not just the creation of knowledge itself. Algorithms expressed in the form of computer programs are more directly beneficial to society than algorithms existing

19. *In re* Gelnovatch, 595 F.2d 32, 36–37 (C.C.P.A. 1979).

20. *See* Patent Act § 101, 35 U.S.C. § 101 (1988); Brenner v. Manson, 383 U.S. 519 (1966).

21. An algorithm can be designed with more than one processor in mind or with no processor in mind but with the intention to refine the algorithm when a particular processor is chosen. However, the algorithm would not be "useful" until the processor is specified. It would be wrong to conclude, in patent doctrine, that anything that was not "useful" was just an idea. As an example, instructions to mix chemicals in a special way could be expressed at a high level, with vague steps such as "stir until mixed." The high-level algorithm could be refined into a more specific algorithm for use by either a machine or a human once the details (as required by each processor) were added. The device may be specified once the high-level algorithm is refined.

solely in academic texts. The algorithms are useful only because a machine can understand and perform such algorithms.[22] No machine as yet can execute algorithms written in natural languages. Thus, what makes a computer algorithm valuable is its form of expression.[23] The difficulty in legally analyzing the exclusion of computer programs from patent protection results from trying to distinguish between algorithms and the implementation of those algorithms in the form of computer programs, where the only significant difference between the two is in the level of detail of expression. Expression is not something the patent system is designed to protect, but it is the valuable aspect of the computer program.

What one must definitely exclude from patent protection is high-level or abstract ideas. Assuming that computer programs are a proper subject matter for patent protection, can a line be drawn within the existing patent framework between a process carried out by a computer program (which is patentable) and the abstract idea that the program embodies (which is not), to determine what computer programs (or, more correctly, which underlying algorithms) warrant patent protection? That line cannot sensibly be drawn between programs and algorithms[24] (as all programs are algorithms)[25] or between useful and non-useful algorithms (as by definition all algorithms are useful)[26] or between laws of nature and algorithms (as no algorithms are laws of nature)[27] or between

22. *See* Maier *supra* note 11, at 151 (Software has functionality that distinguishes it from ordinary writings and "has the power to physically implement [intellectual concepts] with the aid of a computer.").

23. Note that patent law is concerned with determining whether a process is novel and nonobvious and not whether a process is expressed in a more useful way than it has been expressed previously.

24. *See* Bradley J. Hulbert, *Special Considerations for Obtaining and Litigating Software Patents*, 4 SOFTWARE L.J. 1, 3 (1990) (high-level algorithm not computer program).

25. *See* Diamond v. Diehr, 450 U.S. 175, 219 (1981) (Stevens, J., dissenting) (wanting an "unequivocal explanation that the term 'algorithm' as used in this case ... is synonymous with the term 'computer program.'").

26. *See* Newell, *supra* note 1, at 1026 (stating that algorithms are designed to do something useful and that "they jump the gap to application [and therefore are] patentable").

27. An algorithm is not a natural phenomenon or abstract concept. It is a construction of the human mind. Algorithms do not describe natural phenomena. *See* Chisum, *supra* note 11, at 980. However, an algorithm can be expressed at such a high level of abstraction that it is, practically speaking, merely an idea. For example, an algorithm to bake a cake may be "mix ingredients, then cook until brown." Is that an idea or an algorithm giving a high-level description of the solution? It goes without saying that a patent for a process that uses a law of nature, such as a process bottling milk using the law of gravity, does not give the patent holder a patent on the law of gravity.

mathematical and non-mathematical algorithms[28] (as most programs and algorithms are non-mathematical and the distinction would exclude virtually no programs or algorithms from patent coverage).[29] There must be criteria established to determine when an idea expressed as a computer program is sufficiently distinct from the abstract idea that it expresses to warrant patent protection of the process carried out by the program.

However, it would be inconcsistent with the scheme of patent law to prevent the patenting of a process simply because it is expressed in detail—in a form "simple" enough for a computer to understand. As an example, even if a novel way were invented to spray paint a car, the idea of spray painting the car could not be patented. Only that particular novel process could be patented, and only if the patent claim was drafted in enough detail to cover only the process and not the abstract idea. If the process were carried out by a computer, the patent claim could legitimately set out as the process the instructions given to the computer. Failure to protect a narrowly drafted patent claim in the case where a computer is involved has no justification in the language of the patent statute.[30]

When lawyers speak of patenting a computer program (or part of a computer program), what is generally meant is the patenting of the process that the program carries out.[31] No one, at the patent level, is concerned with the protection of the literal code: That is the domain of copyright. It is what lawyers call the process, and what computer scientists call the algorithm, that all the fuss is about. But does granting a

---

28. Chisum, *supra* note 11, at 960, concludes, after deciding that there need to be additional incentives for investment in computer software, that "[n]ew and useful algorithms, including mathematical algorithms, should constitute subject matter eligible for patent protection." Chisum correctly states that algorithms can be devised to solve all sorts of non-mathematical problems. *Id.* at 976.

29. "[B]ecause most software falls in this category of non-mathematical algorithms, a significant percentage of software potentially can be patented." John R. Lastova & Gary M. Hoffman, *Patents: Underutilized Leverage for Protecting and Licensing Software*, 6 COMPUTER LAW. 7, 8 (1989). Mr. Lastova is a Primary Examiner handling computer software applications at the PTO.

30. Differences in levels of abstraction in the description of the algorithm in the patent claim are to be distinguished from levels of invention—a trivial improvement, which is at a lower level of abstraction than a patentable invention or a nonpatentable law of nature, is not patentable and is not what the example is discussing.

31. Patent applications are usually language-independent. The algorithm can be expressed in the form of a flow chart. *See* Maier, *supra* note 11, at 164. To make sufficient disclosure, that flow chart must be able to be used by a programmer of ordinary skill to produce a workable code. It follows that, given the algorithm, any coding of it is obvious. Clearly, the patent holders are trying to protect the algorithm and not any particular coding or form of expression in which the algorithm is expressed.

patent for an algorithm where a computer is the processor in effect protect expression?

The dilemma presented in the above paragraphs can be stated as follows: Patent law does not protect abstract ideas. A computer program is not abstract, but a highly detailed and specific description of a way of solving a problem. However, the underlying algorithm over which the monopoly is sought may be general or abstract. The algorithm in question can be expressed either in a highly abstract way, so that it resembles an idea, or in a highly detailed form such as computer code. The detail of expression gives no indication of the level of abstraction of the process over which patent coverage is claimed. Patent law has difficulties distinguishing between algorithms as ideas and algorithms as processes. If an ad hoc line is drawn excluding all computer programs from patent protection, it excludes algorithms which may be the proper subject of the grant of a process patent. On the other hand, if to determine patentability, a rule is used requiring the processor to be a computer and limiting the scope of protection to the use of the algorithm by the computer, patent law comes very close to protecting expression rather than the underlying process.

At present, the algorithm of any computer program is capable of receiving patent protection provided the algorithm does not simply apply a mathematical formula and provided that the claims are expressed at a level of detail sufficient to distinguish them from abstract ideas. Such protection will prevent any programmer from using a more detailed expression of that algorithm in any computer program. In effect, the patent on the process limits the independent creation of expression.

C. Rationales for Excluding Patent Protection

Many reasons have been suggested as to why computer programs should be excluded from patent protection.[32] Most of these are merely

32. Because of differences in definitions, not everyone has been discussing the same thing. There are at least four possibilities for patent protection involving computer software: A complete program, such as a Computer Aided Design program, that in a new and novel way allows an architect to plan a high-rise building; a section of a program, such as code to store variables in memory efficiently or to locate an item in a database speedily, that performs a computer operation in a new and novel way; a solution (using a computer) to a problem that has not been able to be solved previously, such as a program that determines whether another program is error-free or a program that solves the unemployment problem; and a process that has not been able to be computerized previously, such as a program to keep track of the location of taxis, that uses a computer (and possibly other physical devices) in a new and novel way. *See In re* Abele, 684 F.2d 902, 907 (C.C.P.A. 1982) (A claim that was otherwise statutory, even though less useful without the algorithm, "presents statutory subject matter when the algorithm is included."); Diamond v. Diehr, 450 U.S. 175, 189 (1981) (subject matter must be statutory regardless of presence of computer).

objections to the patent system itself. The most common objections that have been made to the patenting of computer programs are discussed below.

1. In Software, Independent Reinvention Is Commonplace

Compared with other areas of science and "useful arts," it is common for separate computer scientists to "discover" independently the same way of solving a problem.[33] Where independent reinvention is common, one rationale of the patent system, the encouragement of the dissemination of knowledge through publication, loses force: Anyone who considers the problem is likely, without much effort, to arrive at one of a limited number of solutions. As a patent grants a seventeen-year monopoly, anyone wishing to use the process for that period is forbidden to do so, even another independent inventor. Due to the constraints a computer language imposes on the expression of the algorithm, the physical constraints of the computer itself, and the large number of programmers solving problems daily, many argue that there is more repetition of invention than occurs elsewhere in science and business.

In the normal course of events, the subject matter of patent is more general than the subject matter of copyright. Copyright protects expression, where the chance of two people independently producing the same expression is very small. Patent, on the other hand, "operates at the level of generality at which there is at least a plausible possibility of independent creation of the same invention."[34] For this reason, patent has a requirement of novelty, interpreted so as to prevent a patent being granted for an invention that already exists or is obvious to those "skilled in the art." The more abstract the interest for which protection is given, the more likely the odds that two people will independently create the same thing.

For example, there is dispute over whether Newton invented the

33. *See* THE LEAGUE FOR PROGRAMMING FREEDOM, AGAINST SOFTWARE PATENTS 8, unpublished paper of Oct. 24, 1990. The League is "a grass-roots organization of programmers and users opposed to software patents and interface copyrights," *id.* at 1, and includes as members successful entrepreneurs, executives, independent consultants and programmers, including Richard P. Gabriel, John McCarthy, Marvin Minsky, Robert Boyer, and Patrick Winston.

34. THOMAS HEMNES, NOVELTY, SCOPE AND THE SHARED GEOMETRY OF PATENT AND COPYRIGHT PROTECTION 22, paper delivered to the Computer Law Association at Boston, November 5, 1990. *Cf.* John S. Wiley, *Copyright at the School of Patent*, 58 U. CHI. L. REV. 119, 182 (1991), who claims that a partial reason why patent law requires novelty and nonobviousness and copyright does not is because patent innovation is incremental while copyright authorship need not "begin with library research."

system of calculus. Some claim that Gottfried Leibniz came up with the idea first. It is likely, regardless of whether Newton or Leibniz had ever studied mathematics, that someone by now would have discovered and written about calculus. However, it is highly unlikely that anyone would have written a book identical to Newton's *Principia Mathematica*.[35]

In writing computer programs, where the idea is expressed in a highly refined manner, why would the likelihood of independent reinvention be greater? The answer to this depends on how the invention is specified. If the invention is the particular code in a computer program, it is true that once an algorithm has been designed to solve a problem, the likelihood that two independent programmers using the same algorithm and programming language will write the exact same code is quite high. The programmers are merely expressing the algorithm at the level the computer can understand, and are not in fact inventing anything. The constraints of the programming language, if a well defined language, will not allow much room for creative thought process or variations in expression.[36] The likelihood of independent creation of the same program in such circumstances is therefore high.

The inventive steps in software design almost always take place at the level of algorithmic creation. It is the algorithm, and not the code, that the inventor wishes to patent. The question then becomes whether, in the creation of a new and novel algorithm for use on a computer, the likelihood of independent reinvention is greater than where an algorithm or process is created in other fields. Assuming that the level of abstraction is the same both when one is solving a problem where the processor is a computer and where the processor is not a computer, why would the involvement of a computer make independent reinvention of an algorithm more likely?

The League for Programming Freedom, a main proponent of this argument, states generally that a programmer "solves many problems in developing each program. These solutions are likely to be reinvented frequently as other programmers tackle similar problems."[37] This statement is misleading in that it does not differentiate between the creative problem-solving component (the design of the algorithm) and the

---

35. SIR ISAAC NEWTON, PHILOSOPHIAE NATURALIS PRINCIPIA MATHEMATICA (1687).

36. This does not mean, however, that coding an algorithm is mechanical or that all codings of an algorithm will be identical. The point is that the constraints of the language are designed to limit the number of possible ways of expressing a particular algorithm. Compare the process of encoding an algorithm with writing a screenplay from a novel. Although the plot will be the same, the number of possible screenplays will be extremely large.

37. THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33.

mechanical expression of the solution in a language a computer can understand (the coding). As previously stated, it is likely, due to the constraints of programming languages, that there will be similarity in coding. But, is a person who solves a specified problem by designing an algorithm with a computer as the processor constrained in such a way that only a small number of solutions is possible. If so, then the likelihood of independent reinvention is greater where a computer is involved and the rationale for patent protection, to stimulate independent creation, is weaker.

For any problem, there are many conceivable solutions. However, just as to fly between Boston and Washington, D.C. there are many possible routes but only a few sensible flight paths and stopover airports, there exists only a finite number of reasonably efficient algorithms to solve a problem.[38] A finite number does not necessarily mean a number so small that reinvention is likely or that all efficient algorithms will be obvious once the problem is brought to mind. As Professor Newell, a computer scientist, writes, "Algorithms of immense generality and scope will continue to emerge so long as science endures."[39] They will continue to emerge for problems not thought of today and provide better solutions for known problems. Not including simple trivial problems, it would be unreasonable to assume that for all known problems all efficient solutions have been thought of and published. More research will lead to more solutions and better algorithms in many fields. It is a goal of the patent system to encourage such research. That there are a limited number of solutions to a problem does not imply that the first person discovering the solution is not worthy of a patent.

A second concern about the increased probability of independent creation is raised by the existence of computer constraints that limit efficient solutions. Constraints such as available memory limit the size of programs and data, and the silicon chip in the CPU limits the speed of calculation. With today's technology, there may only be one efficient, or even workable, algorithm to solve a given problem.[40] To say that these algorithms are reinvented frequently may just be a restatement that the solutions are obvious to a person experienced in the field. The Patent

---

38. *See* Charles Walter, *Defining the Scope of Software Copyright Protection for Maximum Public Benefit*, 14 RUTGERS COMPUTER & TECH. L. J. 1, 59 (1988).

39. Newell, *supra* note 1, at 1028. If algorithms of importance continue to emerge so long as science endures, why is it necessary for the patent system to encourage such research?

40. A doctrinal response is that there is no rule in patent law that a patent can only be granted to the best invention to solve a problem or will not be granted if the invention is the only solution to the problem.

Act does not, in any event, allow a patent for an obvious invention, process, or improvement. For the argument that independent reinvention is commonplace to succeed, one further premise must be added: There are a small number of nonobvious practical solutions to problems in the field of algorithmic design of computer programs and therefore they are more likely to be reached and rediscovered independently as compared to algorithmic design in other fields. Such a conclusion is difficult to justify. It should not be assumed that designing an algorithm for a computer, as compared to an algorithm designed for any other processor, limits the number of nonobvious solutions to a problem.

Although there may be more computer programmers solving more problems using more accessible technology resulting in greater progress and more inventions (and possibly the same solutions to those multitudes of problems) than in any other field, this phenomenon of intensive creative effort in a developing technology is not new to patent law. At stages of history there is often a race to be first to make a new invention, and as existing knowledge reaches a stage where the time is ripe for such invention, numerous people independently "discover" the same solution.[41] The patent system has previously dealt with such occurrences, maybe not in a very satisfactory way, in its rules about priorities:[42] Only one of the inventors wins the "patent lottery." The criticism here amounts to no more than a criticism of the patent system having to choose between inventors; computer scientists complain that since they invent more, they are more adversely affected by this rule.

It can not be assumed or proven that computer programmers, just by having a computer as the intended processor, are more creative and thus more likely to arrive independently at the same nonobvious solution than would be the case for other inventors. In fact many problems are solved without computers in mind as the processor, but the solution is later used as the basis for a computer program. In that case, invention and any independent reinvention of the algorithm is not affected by having a

41. For example, note the simultaneous discoveries in the glass bottle manufacturing industries, United States v. Hartford-Empire Co., 46 F. Supp. 541 (N.D. Ohio 1942), *modified*, 323 U.S. 386 (1945), the ongoing historical dispute over who invented the airplane, and the 20 year dispute in the Patent Office over who invented the silicon chip.

42. *See* Patent Act § 102(g). 35 U.S.C. § 102(g) (1988). The general rule is, as between two inventors, the first to conceive has priority, so long as the first inventor uses continuous diligence from a time prior to the second inventor's time of conception, and reduces the invention to practice first. The test is ambiguous when applied beyond the two inventor situation. This problem is avoided in other jurisdictions, such as Canada, where the patent is granted to the first to file, not the first to invent.

computer as a processor.[43] This emphasizes the point that an algorithm is not dependent on having a computer as the processor: One would expect the occurrences of independent recreation of algorithms to be the same whether the algorithm is expressed in the form of a computer program or not.

It may be true that independent reinvention is more likely in coding an algorithm, but for algorithmic design, there is no difference in the creative process, indicating that independent reinvention of nonobvious processes is more likely to occur than in other fields.

2. *Most Software Developments Are Nonobvious and Not Novel*

Although there is a distinction between the tests for novelty and nonobviousness,[44] the same attack is made on both—that the tests allow common techniques to be patented, thus hindering legitimate developments in the software industry.[45] It is claimed that the PTO has granted patents on software too easily and without proper knowledge of what is occurring in the software industry.

Prior art must be examined to determine whether the claimed invention is novel and nonobvious. The rapid growth and nature of the software industry has resulted in the commercial success of "back yard" companies. Developments are ad hoc and undocumented. Many new techniques are spread through the use of electronic bulletin boards, where they are not physically stored and where, after a period of time, they are irretrievable or unlocatable.[46] Programmers are more concerned

---

43. The question of whether a person using a known algorithm in a computer program to improve a known, as yet not computerized, process has been the main focus of cases in this area. *See, e.g.*, Diamond v. Diehr, 450 U.S. 175 (1981). The Supreme Court has held that the person doing so has undertaken a sufficient inventive step for the grant of a patent. For processes already in use, there is a strong likelihood that two people will independently try to use that process' algorithm as the basis of a program to computerize the process. If so, the invention is the idea of using a computer to improve a solved problem. That is not a novel or new invention, but an improvement on an old invention. *Cf. id.* at 219 (Stevens, J., dissenting). Justice Stevens wanted "an unequivocal holding that no program-related invention is a patentable process under §101 unless it makes a contribution to the art that is not dependent entirely on the utilization of a computer." *Id.*

44. Simply, novelty depends on what is known, and nonobviousness depends on what the next step may be and if that step has been anticipated.

45. Newell, *supra* note 1, at 1026, states that "it is not possible to do anything in computer science without having it be almost immediately related to use, with only small efforts of the imagination. . . . Hence, where is the rewardable, risky, inventive effort?"

46. Prior art is only relevant if it is exists "in a manner accessible to the public . . . and open to the people of this country . . . upon reasonable inquiry." Galyer v. Wilder, 51 U.S. (10 How.) 476, 496 (1850). The inventor must search with reasonable diligence for materials that could be located by a person ordinarily skilled in the subject matter. Unpublished, uncataloged materials do not, therefore, fall within the category of prior art. *See In re* Hall, 781 F.2d 897, 900 (Fed. Cir. 1986). For computer programs it is likely that most of the

with completing contracts than publishing academic papers: "Sometimes it is possible to patent a technique that is not new precisely because it is obvious—so obvious that no one would have published a paper about it."[47] Determination of what the prior art is or was at any particular time is not the only problem. Matters are exacerbated by the PTO's inexperience in dealing with computer software[48] and its inability to compare patent applications involving computer software with the prior art.[49] This results in patents being incorrectly granted in borderline cases.

A second attack on the current situation is that the "standard of obviousness developed in other fields is inappropriate for software"[50] because the nature of programming encourages the application of techniques used to solve one problem in the solution of a completely separate problem.[51] For example, sub-procedures, such as sorting routines, are used over and over in a variety of programs as a step in accomplishing the ultimate goal of the program. As programmers are trained to generalize, it is obvious to them to use or adapt different techniques to different settings.[52]

Where the programmer is just coding an algorithm, it is difficult to regard the coding of an algorithm as creating something new or involving an inventive step:[53] it is just expressing an algorithm at a different

---

existing prior art will not be able to be found "upon reasonable inquiry" and the Patent Office will grant patents over preexisting processes. *Cf.* Wiley, *supra* note 34, at 142.

47. THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 2–3 ("[M]any commonly-used software techniques do not appear in the scientific literature of computer science. Some seemed too obvious to publish while others seemed insufficiently general; some were open secrets."). *See also* Hulbert, *supra* note 24, at 13, who states that it may be difficult to mount a defense for lack of novelty or for obviousness as "so many of the previously created programs (prior art) may be undocumented, stored only on disk."

48. "The Patent Office refused until recently to hire Computer Science graduates as examiners, and in any case does not offer competitive salaries for the field." THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 2.

49. *See* D. Lee Anton & Theodore A. Feitshans, *Is the United States Automating a Patent Registration System for Software?*, 72 J. PAT. & TRADEMARK OFF. SOC'Y 894 (1990).

50. THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 3.

51. *Cf.* Walter, *supra* note 38, at 84. ("Software develops incrementally; subsequent generations of computer programs are usually obvious and seldom based on novel processes.").

52. Hulbert, *supra* note 24, at 4, claims that under the current law, a patent "may reach far beyond the particular software patented and may relate to other methods that may be useful on far different types of software." It is argued by people opposed to patents for software that such a patent should not have been granted in the first place as in all likelihood it is not a new process but is itself borrowed from elsewhere.

53. Compare the process of translating a patent application from French to English. There is nothing novel or inventive in doing so. As another example, a baker who applies the laws of physics to invent a new way of making bread is rewarded by a patent monopoly when he expresses that process in a patent application and the patent is granted. On the other hand, a computer scientist who develops a successful program may either code an

level of abstraction. If the number of feasible codings is limited by the constraints of the programming language, then it could also be argued that the resulting code is obvious. The automation of the previously manual steps of a process is obvious.[54] Therefore, no patent should be granted to a computer scientist who takes a known algorithm and encodes it in a computer program.

At the higher level of algorithmic design, more complex issues arise. The question shifts to whether or not the "discovery" (and subsequent coding) of a novel and nonobvious algorithm can be distinguished from the simple coding case.[55] As an algorithm may be expressed in many different ways, and used in many disjoint fields, the PTO, as a practical matter, may have limited its inquiry to whether the algorithm in its encoded form is new and nonobvious. If that is the case, then despite the fact that the manifestation of an algorithm as a computer program is always obvious to a computer scientist, what is being protected is the computer code—in other words, expression.

If an idea is directly related to computer programming techniques, such as controlling screen displays or memory management, the algorithms behind such programming "tools" raise similar problems to those raised by simple coding techniques: The algorithm is constrained by the computer's physical construction. However, since the algorithm is not useful where a computer is not involved, the computer becomes both the processor and the object of the process. Most programming tools are likely to be obvious once the the problem surfaces—most innovations occur soon after a new computer is invented.[56] The criticisms that the

---

algorithm already in existence into a form a computer can understand or create a revolutionary new process and code that idea. The second case is more like that of the baker. In the first case, it is difficult to see why the programmer should be given a monopoly—what was done was the creation of a new expression of an old idea, which is the domain of copyright protection. Making a distinction between "translating" an algorithm and "inventing" an algorithm, where both are applied in a useful way, would be a time-consuming and difficult task for a court to undertake.

54. *Cf.* Parker v. Flook, 437 U.S. 584 (1978) (obvious post-solution activity not sufficient to transform an unpatentable principle into a patentable process).

55. *Id.* at 594, setting out the "point of novelty" test: A process is unpatentable if the point of novelty lay in the formula or algorithm in the claim. This case has been described as the "low point for patent protection of software inventions." Maier, *supra* note 11, at 154. The point of novelty test was rejected by the Supreme Court in Diamond v. Diehr, 450 U.S. 175, 189 (1981), which stated that the claim must be considered as a whole.

56. For example, the use of backing store to store the contents of hidden windows was first developed at MIT, on a Lisp Machine. "The Lisp Machine was the first computer to use this technique only because it had a larger memory than earlier machines that had window systems." THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 4. A claim by a computer hardware manufacturer that a programming technique for one of its new machines is new and nonobvious must be regarded with suspicion: The technique may only be new because no one has had the opportunity to write any programs for such a machine and it is likely, if history is any guide, to be obvious to a programmer using that

prior art is not easily discoverable and that programmers regard the flexibility of programming tools as obvious become even stronger when these types of algorithms are considered.

The PTO must be strict in its application of the tests of novelty and nonobviousness to software inventions. Where there is doubt, the safer path in the field of computer programming may be to withhold the patent: the nature of the industry being such that there is a high likelihood that what is claimed is nonobvious and not novel.[57] Secondly, there must be a rule that the mere coding of an existing algorithm or the computerization of a preexisting process (as producing nothing new and only what is obvious) should not lead to the grant of a patent over the algorithm.[58] In other words, no patent should be granted for an algorithm solely due to the fact that it is expressed in a computer language. At what level of abstraction should the PTO and the courts look to see if the algorithm is novel and nonobvious? If the PTO and the courts should not look at the algorithm expressed as code, they must chose a higher level of expression of the algorithm. The higher the level examined, the less likely it it is to be novel and the wider the scope of possible coverage. The closer the level of expression is to computer code, the more the patent system looks as if it is protecting expression. That is the role of copyright. These concerns suggest that patenting is not the optimal form of protection for computer programming innovations.

3. There is No Need for Incentives to Invent in the Software Industry

In 1980, when the law in this area was more uncertain and it was generally believed that there was no patent protection for computer programs, a lawyer wrote that "the industry is growing in leaps and bounds without [copyright or patent protection]."[59] He further suggested that patent protection would stifle competition whereas continuation of the status quo would encourage software developers to improve their pro-

---

machine once he is given the chance to use the new machine.

57. IBM Australia, whose United States parent holds many software patents, submitted to the Australian Copyright Law Review Committee ("ACLR") that "patent law has only a minor role to play in software protection—as most software is insufficiently novel and inventive." Collection of Letters on Copyright Protection of Software 4 (unpublished submissions, on file at the Harvard Law School Library) [hereinafter Collection].

58. "It is not at the coding phase where the primary creativity is expressed in the writing of a computer program." Brief of *Amicus Curiae*, ADAPSO, in Whelan Assocs., Inc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987).

59. Gemignani, *supra* note 4, at 309.

ducts constantly.[60] In 1991, the industry is much more developed and patents have been granted to computer-related processes.[61] No study has been conducted to determine whether the perceived change in the law has enhanced or encouraged innovation or led to an increase in output. If additional incentives are unnecessary, patents will add nothing of value to society, but rather increase costs by granting a monopoly over a useful product or process.

It is not only difficult to quantify the extent to which the patent system encourages innovation that would not otherwise occur;[62] it is a formidable challenge to distinguish the software industry from an unknown general position. This Article merely outlines the arguments that the patent system impedes innovation in software development.

First, it is claimed that there are many other incentives to invent new software apart from the chance of being granted a patent. When there was no patent protection, the industry grew rapidly.[63] Many successful developers never attempted to apply for patents but still produced software for commercial gain or intellectual satisfaction.[64] Innovation, it is claimed, often occurs by accident in solving problems where invention was not the goal of the programmer.[65]

Secondly, it is the market that determines whether an innovation will

---

60. *Id.* at 311.

61. *See, e.g.*, Chisum, *supra* note 11, at 1021–22. The Appendix lists examples of the patents granted, including the technique of using an exclusive-or to write a cursor onto a screen (patent number 4,197,590), a technique to allow several programs to share the same piece of memory (shared copy-on-write segments: patent number 4,742,450), a process for the use of backing store to allow hidden windows to be retrieved quickly (patent number 4,555,775), a menu system for a word processing program (patent number 4,308,582), an algorithm for solving linear programming problems (patent number 4,744,028) and a process for conveying source code into an object program, litigated in Refac Int'l Ltd. v. Lotus Dev. Corp., 131 F.R.D. 56, 57 (S.D.N.Y. 1990).

62. *Cf.* George L. Priest, *What Economists Can Tell Lawyers About Intellectual Property: Comment on Cheung*, 8 RES. L. & ECON. 19, 21 (1986).

63. In 1972, the Supreme Court believed the PTO's claim that there was sufficient growth in the industry in the absence of patent protection. Only copyright protection was then available for computer programs. Gottschalk v. Benson, 409 U.S. 63, 72 (1972). *See also* PRESIDENT'S COMMISSION ON THE PATENT SYSTEM, TO PROMOTE THE PROGRESS OF ... USEFUL ARTS 13 (1966), which recommended denying patentability to software as, without the patent system, the software industry was "doing well." *Gottschalk*, 409 U.S. at 72. The recommendations of the Commission were ignored by Congress.

64. The pace of new development made inventions obsolete quickly, and so it was thought any patent eventually granted would provide no economic return.

65. "Much software innovation comes from programmers solving problems while developing software, not from projects whose specific purpose is to make inventions and obtain patents. In other words, these innovations are byproducts of software development." THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 9.

be successful, not the granting of a patent.[66] Developments occur so swiftly that patented programs are of little commercial value in themselves,[67] and any patent will impede possible improvements that could be made to the program by other developers. The lead time a software developer has in the marketplace gives that developer enough commercial advantage so that the additional incentive of a monopoly over the invention is not needed.[68] However, some software developments can be quickly and cheaply copied and distributed, reducing this lead time advantage. Even when literal copying is not involved, software is different from other products. Software is inexpensive to design and easy to manufacture compared with a hardware system with the same number of components.[69] As software is not designed using real components that have to be physically assembled and tested, and not manufactured in large plants that have to be equipped and tooled, but built from well defined mathematical objects, the reverse engineering of software will take less time than other products.[70]

There is the additional factor that the holder of a patent may license the process. This may be an incentive in itself to create software or to

---

66. Motorola, a computer chip manufacturer lost a key market by delaying the introduction of new products due to "an obsession with technological excellence." Stephen K. Yoder, *Motorola Loses Edge in Microprocessors by Delaying New Chips*, WALL ST. J., Mar. 4, 1991, at A1.

67. "Companies no longer wait for patent authorities to award them unchallengeable first crack at a market. Competition decides who gets the lion's share of the market. Patents, as they arrive, are swapped for royalties or other patents. Instead of being the arbiter of competitive position, patents are becoming just another tradable commodity, like bonds or baseball cards. . . . Given today's shortening product cycles, the ability to create a steady flow of unique, innovative products is far more profitable than trying to stake a claim to any single idea." *The Point of Patents*, ECONOMIST, Sept. 15, 1990, at 19–20. The article presents the example of Mr. Gilbert Hyatt, a self-employed inventor whose patent claim on the microprocessor took 20 years to be granted. If a company waited on such a patent, it would be "irredeemably behind." *Id.* at 20.

See also Tim W. Ferguson, *Liberating Inventors or Shackling Progress with Paperwork?*, WALL ST. J., Mar. 5, 1991, at A17. Mr. Hyatt claims that "a handful of . . . major companies . . . tried to appropriate his work" and that innovators "have been routinely ripped off by lawyer-driven, bureaucratic companies and as a result are holding back breakthroughs that could transform life on earth." *Id.*

68. Even if a process is imitated and marketed speedily by rival firms without the research overheads, competition in that market may be weak so that prices are not driven below a level where development costs cannot be recovered. *Cf.* Stephen Breyer, *The Uneasy Case for Copyright: A Study of Copyright in Books, Photocopies, and Computer Programs*, 84 HARV. L. REV. 281, 345 (1970) (arguing that an imitator will need to develop technical support for copied computer software, thus giving the initial programmer sufficient "lead time" to recover development costs).

69. THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 4.

70. A computer program may still be complex and require a large amount of testing time to see if it performs correctly. An industrial product, with a similar number of components, it is suggested, would be far more complex to design and test. *Id.* at 5.

acquire patents on software that others have been induced to create. In the hardware field, revenues from license fees have been high. For example, Texas Instruments Inc. in 1987 decided to raise the license fee on its chips to five percent, generating 281 million dollars of income in two years from protesting rivals.[71]

Thirdly, software is often designed by universities as part of research or through government subsidized programs.[72] The resulting inventions are not due to the incentives of the patent system. On the other hand, universities often patent and develop the results of research, and the investment by private enterprise in such ventures would be unlikely without the knowledge that the product or process was patented.[73] A reply may be that such institutions are inefficient in their development of marketable software and that the large amounts of money spent take resources away from more socially desirable or efficient projects.

Without further study and economic analysis of the patent system, no final solutions can be reached. The issues that computer software innovation present, due to the infancy of the industry and the different methods of production used, will put a gloss on any general findings about the patent system. In the long term, there is no reason to believe that the ratio of innovation in the software industry because of the patent system to innovation in spite of the patent system will be different from that in other industries.

At present, the expression in software is protected by copyright. Whether patent should provide protection to programs in addition or as an alternative to copyright is examined in the Conclusion. However, due

---

71. Paula Dwyer et al., *The Battle Raging over "Intellectual Property,"* BUS. WK., May 22, 1989, at 79.

72. An example is ADA, a computer language designed by the U.S. Defense Department.

73. For example, the British Technology Group ("BTG") claims to be "the world's leading technology transfer organization, licensing and financing products worth over 600 million pounds in annual sales." Universities, polytechnics, and government research establishments in the U.K. are BTG's most important inventive sources. *See* BRITISH TECHNOLOGY GROUP, THE WORLD'S LEADING TECHNOLOGY TRANSFER ORGANIZATION, firm brochure, undated. The point is that the academic may invent regardless of incentives, but the financier will not risk the possibility of imitation by rival firms when deciding whether to invest in further development and construction. The countervailing arguments are that not all socially valuable inventions are given patent monopolies and are still developed, and that research by universities or government subsidized bodies is inefficient, producing only a tiny fraction of worthwhile products compared with the amounts invested. BTG's promotional materials do not say what percentage of their patent portfolio consists of software patents, and state that for some inventions copyright is the preferred form of protection. It would be useful to know if companies readily invest in software projects where the only intellectual property protection is copyright. BTG recently opened an office in the United States.

to the fact that software can be copied easily, one may assume that the incentive to write programs would be less if no protection were provided at all. This Section has considered only whether protection of computer-related algorithms by the patent system encourages or stifles program creation. One should note in conclusion that since it is only recently that the PTO has granted patents on software, most innovation in the industry has not occurred because of patent protection.[74]

4. Licensing of Software Patents Does Not Work

There are three common arguments why licensing of computer software inventions is harmful to the software industry and the community. Although these may be valid complaints, this Section will only show that they are not unique to software patents: It will be left for others to prove (if it is possible) that the patent system does not fulfill its claim of promoting efficiency by enabling others to license existing inventions.

The first argument is that patents of computer programs are hard to find, and if found, are impossible to understand. The PTO has no classification for software or computer related inventions. They are filed everywhere and anywhere, "most frequently classified by end results . . . but many patents cover algorithms whose use in a program is entirely independent of the purpose of the program."[75] Even a diligent inventor who did not wish to infringe another's patent might be unable to find out whether a patent exists over a certain process.[76]

When a possible patent is found, even though it is meant to disclose to the world the new invention or process, it is often difficult or impossible to understand. Another computer scientist, reading a patent, would have problems in establishing if the patent covered his or her invention, and as the patent owner is likely to assert a wider patent than actually exists,[77]

---

74. *See generally* Hans A. von Spakovsky et al., *The Limited Patenting of Computer Software: A Proposed Statutory Approach*, 16 CUMB. L. REV. 27, 44–45 (1986) (noting that the computer industry has grown in the absence of patent protection and suggesting that such protection may act as a disincentive for innovation). In addition, von Spakovsky et al. claim that patenting software would encourage mediocrity. "The first new nonobvious program performing a particular function . . . would be patentable . . . regardless of . . . how efficiently it ran." *Id.* at 45.

75. THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 6. *See also* Gottschalk v. Benson, 409 U.S. 63, 72 (1972) (citing report of the President's Commission on the Patent System which stated that the PTO could not patent computer software due to a lack of classification).

76. The answer to this argument is not to prohibit the patenting of computer programs and algorithms, but to provide a comprehensive and easily searched register of such patents.

77. *But cf.* Walker Process Equip. Inc. v. Food Mach. & Chem. Corp., 382 U.S. 172 (1965) (Maintaining and attempting to enforce a patent procured by fraud may itself violate the Sherman Act and entitle the injured party to recover treble damages.).

would be hesitant to proceed with possible conflicting works. On the other hand, how can a judge, not technically trained, be expected to deal with issues involving computer patents? Such complaints are not new or limited to computer software patents.[78]

Secondly, as most commercial programs are large and developed using many software techniques and algorithms, an inventor of new software will be unable to search the patent register for every process used in the new program since each patent search costs over a thousand dollars and the new program may have thousands of danger points. If a royalty had to be paid to each patent holder, the marketing of the program would be unprofitable. For example, if a program contained twenty previously patented inventions, each licensed at a rate of one percent, the second programmer would be at a large commercial disadvantage breaking into the market. It would be worse if the program contained one hundred patented processes, which is possible for large programs if patents continue to be granted by the Patent Office for computer software.

This leads into the third complaint—that existing firms can stifle competition, and therefore innovation, by obtaining licenses over many different inventions, and keep rivals out of the market by refusing to license, by charging excessive license fees, or by forcing rivals to waste resources inventing inferior processes that accomplish similar results in less efficient ways.[79] The problem is worse for small time programmers, as many existing patents are invalid and will be declared so if tested in the courts.[80] These programmers do not have the money or legal

78. *E.g.*, Nyyssonen v. Bendix Corp., 342 F.2d 531, 533–34 (1965) ("[W]e cannot read [the patents] intelligently. . . . Moreover, we have great difficulty in understanding, even in a general way, the technical testimony and the discussion of that testimony by counsel.").

79. A further method of reducing competition is the threat of high damages in patent suits, which some authorities have claimed "makes the patent system a 'tool of extortion'" with claims up to 3.3 billion dollars. Joseph M. Fitzpatrick & Robert H. Fischer, *Patent Damages*, ELECTRONIC AND COMPUTER PATENT LAW 737, 760 (Practicing Law Inst. Course Handbook Series No. 292, 1990). With regard to determining damages, the authors state that as "software patent claims typically must include limitations and/or steps in addition to the algorithm in order to claim patentable subject matter . . . it is possible to imagine a royalty base for a patented software program including a computer selling at tens or hundreds of times the price of the program." *Id.* at 759.

80. During the 1960s, fewer than 40% of patents were upheld by the courts. *See* IRVING KAYTON, THE CRISIS OF LAW IN PATENTS, Table A-2 (1970). *See also* PHILLIP AREEDA & LOUIS KAPLOW, ANTITRUST ANALYSIS, PROBLEMS, TEXT, CASES ¶ 186(d) (4th ed. 1988) (The PTO "often seems to resolve doubts about patentability in favor of issuance," especially in close cases so that the examiner's decision is not appealed.). However, these criticisms apply to the patent system generally, and not just in relation to computer software patents.

resources to challenge the big software firms.[81]

The problem is worse when it is considered that the holder of a patent may have little incentive to license software to rivals. Patent protection allows a software developer to introduce a program into the market without having to license it and expand the network of rivals. A firm with brand recognition thus would reap increased rewards by preventing a rival reducing the well recognized firm's market share. A compulsory license may be the solution to this problem, especially considering that society benefits from the expanded network. The purpose of such a compulsory license is said to be to reduce the extent to which patent ownership of the process conveys monopoly power.[82]

These "abuses" of patents, if the strict enforcement of a government granted monopoly can be regarded as an abuse, are not new or solely software related. For example, the United Shoe Machinery Corporation, in the 1950s, had 3,915 patents that, to some extent, blocked potential competition in the shoe-making business.[83] Where patents are abused to create monopolies or to restrain trade, the antitrust laws may provide the desired remedy.[84] Further, a small firm may be granted a patent for its

---

81. Worse still, the big firms may use the threat of the (possibly invalid) patent to close down or purchase a rival's business. *See, e.g.*, Hartford-Empire Co. v. United States, 323 U.S. 386 (1944), and discussion of the case in AREEDA & KAPLOW, *supra* note 80, at 463–67. However, the engineers that formed the Hartford Company did so only to profit from the patent system and were not interested in glass manufacture as an example of the patent system encouraging innovation. *Cf.* United States v. General Elec. Co., 272 U.S. 476 (1926) (Holder of a valid patent, subject to continuous legal challenges and infringements, decided that the easiest way to control the industry/market was to license all applicants and infringers.). *See also* AREEDA & KAPLOW, *supra* note 80, at 428 n.2.

82. *See* Jane C. Ginsburg, *Creation and Commercial Value: Copyright Protection of Works of Information*, 90 COLUM. L. REV. 1865, 1924–27 (1990).

83. United States v. United Shoe Mach. Corp., 110 F. Supp. 295, 333 (D. Mass. 1953), *aff'd per curiam*, 347 U.S. 521 (1954). The court, in compelling the defendant to grant any applicant a nonexclusive license at a "reasonable royalty" under any patent now or subsequently acquired from a nonemployee, stated that the defendant was "not being punished for abusive practices respecting patents." *Id.* at 351.

84. *See, e.g.*, United States v. United States Gypsum, 340 U.S. 76 (1950); Transparent-Wrap Mach. Corp. v. Stokes & Smith Co., 329 U.S. 637 (1947); United States v. General Elec. Co., 82 F. Supp. 753 (D.N.J. 1949). *But see* SCM Corp. v. Xerox Corp., 645 F.2d 1195 (2d Cir. 1981), *cert. denied*, 455 U.S. 1016 (1982); Louis Kaplow, *The Patent-Antitrust Intersection: A Reappraisal*, 97 HARV. L. REV. 1813 (1984); Lasercomb Am., Inc. v. Reynolds, 911 F.2d 970 (4th Cir. 1990) (anticompetitive language in software program license amounted to misuse of copyright and barred infringement even if misuse was not antitrust violation); *Clarifying the Copyright Misuse Defense: The Role of Antitrust Standards and First Amendment Values*, 104 HARV. L. REV. 1289, 1299 (1991) ("[W]hether copyrighted computer programs are likely to enjoy market power—and thus whether a finding of misuse based on antitrust standards is more likely—will depend on how the courts define the scope of protection for computer software.").

invention and so have the ability to enter a concentrated market successfully and compete where it otherwise would not be able to do so.

5. Software Developments Build on Previous Developments

Large software projects are often built from the components of other programs and use techniques developed for other applications but modified or adapted to fulfill the new project's goals.[85] Novel successful programs can be developed from similar less successful programs, so as to add a more congenial user interface, or to add new features,[86] or to run on more popular machines or with more accessible operating systems,[87] or to run faster[88] or to use less memory: The underlying algorithm is the same but other features of the program are enhanced.[89]

If patents were granted for the the underlying algorithm, these advances, which often make the program commercially successful, would not be permitted to occur without the permission of the patent holder. Inventors may be deterred from further research by the realization that improvements could not be made without infringing the original patent. This may lead to doubly wasteful results: The original invention may, although superior, be ignored in the marketplace, and the new inventor will, to compete, have to waste resources inventing around the original patented algorithm.

Consumers invest large amounts of money in particular software systems, both in purchase costs and, more particularly, in staff training costs. One reason that consumers so invest is that they are promised that the software will be upgraded and enhanced by the manufacturer: The consumer will always have the best software without having to repur-

85. In fact, most new useful inventions are based on products produced from original inventions, and not on the original invention itself. *See* Newell, *supra* note 1, at 1034.

86. *E.g.*, a spelling check program is used in conjunction with a word processor that modifies the screen display by adding another pull-down menu.

87. *Cf.* Richard H. Stern, *Copyright Infringement by Add-on Software*, 31 JURIMETRICS J. 205 (1991) (The program "Masquerade" makes programs written for IBM mainframes appear to be written for the Macintosh.).

88. *Cf.* Artic Int'l Inc. v. Midway Mfg. Co., 704 F.2d 1009 (7th Cir.) (involving a speed up kit for a video game), *cert. denied*, 464 U.S. 823 (1983).

89. This problem is different from the conventional improvement problem in that the underlying algorithm may be used when "adding-on" features to existing programs. An add-on feature could be a separate program that runs concurrently with the main program, or modifies the code of the main program when running, or it could incorporate the main program and the add-on in the one program presale. Possibly only the last example would infringe the patent of the algorithm of the main program. Alternatively, the new program could take a successful feature from an existing program and add it to the new program, thus using the algorithm in a different context or in a superior way (for example, with a more useable interface).

chase new systems or retrain staff each time a technological improvement is made. If the firm producing the software is either infringing an existing patent or unable to upgrade due to a patent on the "enhanced" process, the consumer will lose. The choice will be to remain with the outdated software or retrain staff on new software. This software may itself become outdated and the company manufacturing it may be prevented by another patent from enhancing that software.

The policy question that arises is whether society will benefit more by providing a wider scope for protection for the original inventor, and thus encouraging that inventor to risk more due to the possible rewards of the licensing of enhancements, or by encouraging subsequent inventors to improve existing software. The question is "whether allocating the incremental value of the new technological use to the original or to the subsequent entrepreneur will lead to more creation and marketing of technological advances."[90] It is a question that can be asked for all derivative works and is examined in detail elsewhere.[91]

D. Conclusion on Patents

The criticisms presented above do not indicate a clear answer to the question of whether patent protection should be denied to computer programs. The criticisms, when examined closely, either present old problems in a new form or, with sensible arguments both ways, leave a policy choice to be made. The consequences of such a choice could have a long-term effect on the software industry.

It has been suggested that, as the pre-patent software industry had "no problem that was solved by patents" there should be a complete elimination of software patents. As the answers to the questions presented above are not clear, the former simple position—granting no software patents—should be adopted in case the answers to the questions turn out

90. Stern, *supra* note 87, at 212. The author concludes that "society gains more from rewarding the subsequent entrepreneur than the original one." The original entrepreneur did not foresee or market the new technology and its possible existence was not an incentive to produce to that person. The subsequent entrepreneur will have little incentive to develop an improvement if the law could be used to shut him down. *See also* William M. Landes & Richard A. Posner, *An Economic Analysis of Copyright Law*, 18 J. LEGAL STUD. 325, 332 (1989) (more extensive copyright protection would raise cost of creating new works and reduce number of works created).

91. *See* Commission of European Communities, *Green Paper on Copyright*, at 172, ¶5.2.9, June 7, 1988 (stating that the real profit for the software house is in adding value to the original software by adapting it for each user). *See* Robert P. Merges & Richard R. Nelson, *On the Complex Economics of Patent Scope* 90 COLUM. L. REV. 839 (1990).

to be against the protection of software developments by patents.[92]

A second solution is to clarify the position to definitely allow patents for software, like other processes, pending a detailed review of the patent system. If a process is otherwise patentable, it should be irrelevant that a computer is the intended processor. If an algorithm implemented in hardware is patentable, why should it not be patentable if it is implemented in software?

To review, a computer program is a detailed and precise expression of an algorithm. An algorithm is a set of instructions to carry out a process,[93] and processes that are novel and nonobvious are patentable.[94] When one talks of patenting a computer program, it is not the expression that is the subject of the patent, but the underlying algorithm. The expression is just that—a way of articulating the algorithm so that someone or something, a programmer, a patent examiner, or a computer, can comprehend the algorithm.[95] Thus, to follow the argument through, all computer programs that express a novel and nonobvious algorithm should be patentable, or rather, the patent should be granted over the algorithm represented by the program. Anyone desiring to carry out that process in that way can be prohibited. It is irrelevant that a computer is used as the processor.

At this point one may conclude that all novel and nonobvious algorithms implemented in a useful way with a computer processor should be granted patent protection. A nonexclusive test to determine patentability of subject matter would be easy: Can a computer understand and carry out the algorithm? If so, then the algorithm is expressed at a sufficient level of detail and is patentable subject matter.

---

92. *See* THE LEAGUE FOR PROGRAMMING FREEDOM, *supra* note 33, at 9. ("If it is ever shown that software patents are beneficial in exceptional cases, the law can be changed again at that time—if it is important enough. There is no reason to continue the present catastrophic situation until that day."). The League would only allow patents for "implementations in the form of hard-to-design hardware" but not implementations of patented processes in software. *Id.* at 10. Simply, the distinction they propose is between algorithms implemented in hardware and algorithms implemented in software.

93. *See* Chisum, *supra* note 11, at 975 (The current definitions come close to equating algorithm with a "process in the patent law sense of a sequence of specifically defined operations that accomplish a useful result.").

94. *See* Newell, *supra* note 1, at 1031 (After stating that all inventions, including the transformation of matter, may occur by the invention of algorithms, the author concluded, "[i]f methods and processes over large technological domains become an exercise in algorithms, then it is extraordinarily dangerous not to patent algorithms.").

95. *Cf.* Wiley, *supra* note 34, at 123. ("[A]n idea inevitably becomes a concrete expression as soon as a human states it. That is, an idea cannot be defined or communicated to another person without becoming an expression, a particular and precise collection of meaningful symbols."). An "idea" communicated to a computer in the form of a program could not therefore be an abstract idea according to Wiley's logic.

However, this proposed subject matter test leads to difficulties. As computer science progresses, computers will be able to carry out more processes and understand higher levels of expression, even possibly natural languages such as English. The scope of protection would therefore increase over time to include algorithms expressed in more vague and general ways. The more general the expression of the algorithm, the more applications that the algorithm covers.

Secondly, people would code an algorithm just to get patent protection over the algorithm. An inventor of a process which may otherwise be unpatentable as an abstract idea would need only to code it to have a patent over the process. The resulting computer program itself would be of little use.[96] If the scope of the patent was limited to applications where a computer was the processor, then other inventors would be free to build machines that were not computers that used the algorithm to accomplish the same results without infringing the process patent.

The proposed subject matter test in reality only determines if the level of expression of the algorithm is sufficiently detailed to give the algorithm patent protection.[97] But why should a subject matter test for patent law say anything about levels of expression? The point of the subject matter test in relation to processes is to make sure that the process is not particularly abstract. An abstract process is barred from patent protection because, although the inventor was the first to articulate the methodology, it is so basic a process that it could be regarded as naturally occurring. An abstract process is unpatentable because it effectively

96. A related problem is that if computer programs were patentable, due to the large numbers written daily, the costs involved in administering the patent examination system would be burdensome. With many applications in which fine distinctions had to be made, the likelihood of erroneous decisions (and the resulting costs to society) would be high.

97. If the patent application describes the program at a high level of expression, then as a consequence it could be regarded as too wide in scope to catch programs expressed in lower level languages where the detail of expression has changed and the processor has become less sophisticated. Alternatively, the patent on the algorithm may cover the use of that algorithm in all programming languages in which that algorithm could be expressed, regardless of the form of expression, prohibiting the use of any program in any language performing that task in that way. *See* Gottschalk v. Benson, 409 U.S. 63, 72 (1972) (The patent "would wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself."). The middle ground is that it only catches programs written using that algorithm in the language in which the patent application is expressed. This would be close to what copyright protects: the form of expression.

To comply with disclosure requirements, the patent application must describe the process in detail to enable an ordinary skilled programmer to draft a workable code with no more than a reasonable degree of difficulty. Disclosure is usually by flow diagrams, which are language-independent. *See* Maier, *supra* note 11, at 164. All programs, in whatever language, using that algorithm, would be within the scope of such a patent. Alternatively, if the patent application disclosed the exact code, it might have a "bug" in it, and lead to the argument that the disclosure is not enabling or is of a useless process.

includes too much within its scope. The proposed test mistakenly looks at the level of expression of the algorithm, not the level of abstraction.[98] For example, a new algorithm to find the average of three numbers is regarded as too abstract for patent protection even though it can be expressed easily in computer code. If, to prevent the patenting of abstract ideas, the algorithm were limited to where a computer was the desired processor, what in fact was protected would be the algorithm expressed in the form of a computer program. If copyright protects the expression of algorithms there is no need for patent to do the same.

As has been seen previously, the patent system should not give protection to the coding of a preexisting algorithm or the computerization of a preexisting process. What would be left for the patent system to protect and encourage is the design of novel and nonobvious algorithms. The courts would have a difficult task deciding into which of these classes a program/algorithm falls. One cannot tell just by looking at the code alone whether the programmer coded an existing algorithm or created a novel nonobvious algorithm.[99] At the end of the day, there may be few algorithms, where the computer is the desired processor, that pass such tests. One doubts whether the patent system is in the best position to make such determinations.

Whether the intellectual effort of algorithmic design should be protected by the grant of a patent cannot be decided without first examining the scope of the copyright regime. For if copyright protects all that is worthwhile without any of the bad side effects that it is claimed that the patent system has, it would be inefficient to give protection under both regimes.

---

98. Patent law says to ignore the level of expression of the algorithm and examine only the algorithm. It is a philosophically difficult problem to ignore the expression of something that only exists once expressed. Is an algorithm that carries out the same process in the same way but expressed at a different level of abstraction the same algorithm after all? Can one distinguish the dancer from the dance?

99. A court would also have to determine whether different codings were of the same algorithm. The court would have to look in detail into the different programming structures and constructs and understand how both the algorithm and programming language worked. A court could not look only at the result or output of the program. Different algorithms can be used to produce the same results. Therefore, Maier, *supra* note 11, at 158, is overbroad in saying that a patent protects "the functional aspects of software." *Cf.* Pursche v. Atlas Scraper & Eng. Co., 300 F.2d 467, 482 (9th Cir. 1961), (The alleged infringing invention must have substantial identity of function, means, and results), *cert. denied*, 371 U.S. 911 (1962).

III. COPYRIGHT

Copyright protection, like patent protection, exists on the theory that "the public benefits from the creative activities of authors; and that the copyright monopoly is a necessary condition [for] such creative activities."[100] Copyright does not protect an idea alone, but the tangible expression of an idea is protected, provided that that expression constitutes "the fruits of intellectual labor"[101] and has not itself been copied from elsewhere.[102] An algorithm can be expressed in the form of a computer program. Copyright will prevent, at the least, literal copying of the computer program. However, as the algorithm of the program can be expressed in different languages and at different levels of abstraction, it is not clear to what extent copyright prevents others from expressing that algorithm in different ways.

A. The Scope of Protection

Copyright protects more than the literal expression. The plot of a novel,[103] the characters in a movie,[104] and the melody of a song[105] are all protected. A computer program is presently regarded as a literary work.[106] Thus, unless there are reasons to the contrary, the nonliteral elements of a computer program should be protected as well. This is in fact the way the law has progressed in the United States: Courts have protected a program's structure, sequence and organization,[107] user

---

100. MELVILLE B. NIMMER & DAVID NIMMER, 1 NIMMER ON COPYRIGHT § 1.03[A] (1990 & Supp. 1991).

101. The Trademark Cases, 100 U.S. 82, 94 (1879).

102. See NIMMER & NIMMER, *supra* note 100, § 1.08[C].

103. *See, e.g.*, Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (2d Cir. 1930); Holland v. Vivian Damon Prods. [1926–45] MacG. Cop. Cas. 69 (ChD).

104. *See, e.g.*, Ideal Toy Corp. v. Kenner Prods. Div., 443 F. Supp. 291 (S.D.N.Y. 1977); Sid & Marty Krofft Television v. McDonald's Corp., 562 F.2d 1157 (9th Cir. 1977); Warner Bros. v. Columbia Broadcasting Sys., 216 F.2d 945 (9th Cir. 1954); Walt Disney Prods. v. Air Pirates, 581 F.2d 751 (9th Cir. 1978); *cf.* Kelly Cinema Houses [1928–35] MacG. Cop. Cas. 362 (ChD).

105. *See* Bright Tunes Music Corp. v. Harrisongs Music, 420 F. Supp. 177 (S.D.N.Y. 1976).

106. In the United States, *see* H.R. REP. NO. 1476, 94th Cong., 2d Sess. 54 (1976); Whelan Assocs. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1243 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987). In Australia, *see* Copyright Act § 10 definition of literary works that includes "(b) a computer program or a compilation of computer programs." *See also Dyason v. Autodesk, Inc.*, 96 A.L.R. 57, 83 (Full Federal Court 1990).

107. *See Whelan*, 797 F.2d at 1240, 1248; SAS Inst. v. S & H Computer Sys., 605 F. Supp. 816, 830 (M.D. Tenn. 1985); Johnson Controls v. Phoenix Control Sys., 706 F.2d 1173, 1175 (9th Cir. 1989); Telemarketing Resources v. Symantec Corp., 12 U.S.P.Q.2d (BNA) 1991, 1993 (N.D. Cal. 1989) (holding that plaintiffs may not claim copyright protection of expression that is, if not standard, then commonplace in the software industry).

interface,[108] and screen displays.[109]

1. Expressions Are Protected, Not Ideas

Given that the idea-expression distinction is the fundamental test in copyright law to determine the scope of protection,[110] computer programs and algorithms will be analyzed within that framework to see if consistent and practical rules can be devised to decide what forms of expression should be protected.

The current state of the law in this area will be examined in the jurisdictions of the United States and Australia. Both are common-law countries that have the same basic copyright framework embracing the idea-expression distinction.[111] The copyright legislation[112] in both countries has been enacted by the federal lawmaking body under somewhat similar grants of power in their respective constitutions.[113] Both countries are members of the Berne Copyright Convention. However, the copyright laws of Australia and the United States reflect differences of analysis that affect computer software copyright. There is no policy reason why the protection provided should differ, since the same software is sold in these countries and they have similar laws. This Article, after disposing of two unhelpful doctrines, will propose tests that can be used in both jurisdictions to produce sensible results consistent with the policy of copyright law.

2. The State of the Law

The law in this area is not stable. In Australia, the Copyright Law Review Committee, under a reference from the Attorney General, is

---

Contra Plains Cotton Coop. v. Goodpasture Computer Servs., 807 F.2d 1256, 1262 (5th Cir. 1987).

108. Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37 (D. Mass. 1990). One of the inventors of Lotus 1-2-3, Mitch Kapor, referring to the *Lotus* case at a forum at MIT, *Intellectual Protection of Software*, Oct. 30, 1990, said "I sometimes feel like Dr. Frankenstein." Kapor is a software minimalist, wanting protection only of the literal elements in the source and object code.

109. Manufacturers Technologies, Inc. v. CAMS, Inc., 706 F. Supp. 984, 993 (D. Conn. 1989); *Telemarketing Resources*, 12 U.S.P.Q.2d (BNA) at 1993; Broaderbund Software, Inc. v. Unison World, Inc., 648 F. Supp. 1127, 1133 (N.D. Cal. 1986).

110. *See* NIMMER & NIMMER, *supra* note 100, § 1.03[D].

111. *See* Blackie & Sons Ltd. v. The Lothian Book Publishing Co. Pty. Ltd. 29 C.L.R. 396, 400 (1921); Hollinrake v. Truswell, 3 Ch. 420 (1894); Baker v. Selden, 101 U.S. 99 (1879).

112. U.S.: 17 U.S.C. (1976); Aust: Copyright Act 1968.

113. U.S.: U.S. CONST. art. I, § 8, cl. 8; Aust: AUST. CONST. § 51(xviii).

currently deciding whether the existing Copyright Act "adequately and appropriately protects computer programs."[114] The High Court of Australia has granted special leave to appeal the leading Australian computer copyright case, *Dyason v. Autodesk Inc.*[115] This case provides an illustration of the difficulties involved in applying copyright protection to computer programs.

In the United States, the protection of user interfaces is strongly contested by various members of the software industry.[116] One side is concentrating its efforts on expanding the scope of protection through court action,[117] while the opposing side is lobbying for legislative changes to limit copyright protection of user interfaces.

B. Two Unhelpful Doctrines

In cases involving infringement of copyright in computer programs, the United States courts have used two doctrines to help resolve the difficult issues that have arisen. The doctrines are "merger of idea and expression" and "no protection for useful articles." This Section will show that these doctrines are unhelpful in computer program copyright cases.

1. Useful Articles

In the United States, purely utilitarian objects are not subject to copyright protection; the utilitarian aspects of useful articles[118] are not works of authorship in which copyright can exist.

---

114. Reference of Attorney General of Australia, Oct. 1988, to Australian Copyright Law Review Committee.

115. 96 A.L.R. 57 (1990) (Full Federal Court decision: Lockhart, Sheppard, & Beaumont, JJ.), special leave granted Nov. 16, 1990 (Mason C.J., Gaudron & McHugh, JJ.). Sheppard, J., is Chairman of the Copyright Law Review Committee.

116. A computer user must communicate with the program and the program must communicate with the user. This communication is via what computer scientists call a user interface. The interface is partially expressed as part of the program's output, usually on the computer's screen. Should the law prevent the copying of a program's interface given that no part of the computer program producing the output and using the interface has been copied?

117. *See, e.g.*, Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37 (D. Mass. 1990); Xerox Corp. v. Apple Computer, Inc., 734 F. Supp. 1542 (N.D. Cal. 1990); Apple Computer, Inc. v. Microsoft Corp., 759 F. Supp. 1444 (N.D. Cal. 1991); Michael B. Bixby, *Synthesis and Originality in Computer Screen Displays and User Interfaces*, 27 WILLAMETTE L. REV. 31 (1991).

118. "A 'useful article' is an article having an intrinsic utilitarian function that is not merely to portray the appearance of the article or to convey information." 17 U.S.C. § 101 (1988).

The United States Copyright Act gives copyright protection to works of authorship, including "pictorial, graphic, and sculptural works."[119] Such works are defined as including "works of artistic craftsmanship insofar as their form but not their mechanical or utilitarian aspects are concerned."[120] The artistic features must be able to exist independently of, and be identified separately from, the utilitarian aspects of the article.

The scope of exclusive rights in pictorial, graphic, and sculptural works is limited by section 113(b). This subsection refers back to the law existing prior to the commencement of the 1976 Act: Simply, the copyright in a pictorial, graphic, or sculptural work, portraying a useful article as such, does not extend to the manufacture of the useful article itself. The owner of the copyright in a drawing has no copyright over the useful article portrayed in the drawing. A useful article built from a two-dimensional drawing does not infringe the copyright in the drawing.

Second, a three-dimensional object only has copyright protection for its form; there is no copyright protection for any useful feature of the work. If the article is purely utilitarian it has no copyright protection.[121] A three-dimensional object which is an architectual work, however, is protected by copyright even though it is useful.[122]

The leading case prior to the 1976 Act is *Maizer v. Stein*,[123] where the Supreme Court held that works of art that had been incorporated into the designs of useful articles were copyrightable. In that case, an artistic female Balinese dancer statuette was used as a lamp base. It was irrelevant that the artist had the intention to mass-produce the design as part of a commercial article and that the design lacked aesthetic value. Congress enacted the basic rule of *Maizer v. Stein* in the 1976 Copyright Act.[124]

Later cases have interpreted section 113(b) as requiring the functional concerns of the article to have no influence on the work's aesthetically pleasing appearance.[125] There is no copyright protection if the form of

---

119. *Id.* § 102(a)(5).

120. *Id.* § 101.

121. *Cf.* Talk of the Town Pty. Ltd. v. Hagstrom, 99 A.L.R. 130 (1991).

122. An, as yet unenacted, bill would amend the law to allow copyright of architectural works. Copyright Amendments Act of 1990, H.R. 5498, 101st Cong., 2d Sess. (1990). The old law is stated in Demetriades v. Kaufmann, 680 F. Supp. 658 (S.D.N.Y. 1988); David E. Shipley, *Copyright Protection for Architectural Works*, 37 S.C. L. Rev. 393 (1986).

123. 347 U.S. 201 (1954).

124. 17 U.S.C. § 113 (1976).

125. *See, e.g.*, Brandir Int'l v. Cascade Pac. Lumber Co., 834 F.2d 1142 (2d Cir. 1987) (form of an undulating tube bicycle rack inseparable from its function).

the artistic work is dictated by its utilitarian function.[126] The artistic elements of the work must be physically[127] or conceptually separable from the utilitarian aspects of the work.[128] Where the design elements can be identified as reflecting the designer's artistic judgment exercised independently of functional influences, conceptual separability exists, and the artistic elements are copyrightable. Copyrightable art does not lose its copyright just because it is put to a function, but if the design of the art is changed to make it more functional, then the cases hold that copyright protection ceases.

Computer programs that contain no errors are useful. It is not correct to say that useful works are not subject to copyright protection: Maps[129] and music, both useful in the same sense that computer programs are useful, are clearly the proper subject matter of copyright and have been so since 1790 and 1831 respectively. An argument, which is constantly made in this area,[130] is that a computer program is a useful article and so has no copyright protection. This argument misunderstands the useful article doctrine; computer programs are not "pictorial, graphic or sculptural works." Even works that are functional, such as houses, are given copyright protection.

A more refined argument is that a screen display is a useful pictorial or graphic work, or is part of a useful work, and is therefore not protected by the Copyright Act. However, the computer display is not a useful article made from an artistic work. The computer program is not an artistic work. A screen display can not be regarded as being made from a representation of the screen in the computer program in the same

126. A jump rope, a billiard ball, and a contact lens are examples of items where the function dictates the form.

127. For example, a sculpture attached to the front of a boat is separable from the utilitarian function of the boat.

128. *See* Carol Barnhart Inc. v. Economy Cover Corp., 773 F.2d 411 (2d Cir. 1985) (clothes mannequins not copyrightable since artistic features are inseparable from use; it is irrelevant that the mannequin is pleasant to look at.); Keiselstein-Cord v. Accessories by Pearl, Inc., 632 F.2d 989 (2d Cir. 1980) (belt buckle made out of sculptural design is copyrightable).

129. The definition in 17 U.S.C. § 101 of "Pictorial, graphic and sculptural works" specifically includes maps, and by inference, does not include maps within the later term "works of artistic craftsmanship," so a map *would* be copyrightable, regardless of whether the graphic features can be identified separately from the utilitarian aspects. A map is not a "useful article" as it must have an intrinsic function other than to convey information itself.

130. *Contra* Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37, 58 (D. Mass. 1990) ("A more sensible interpretation of the statutory mandate is that the mere fact that an intellectual work is useful or functional—be it a dictionary, directory, map, book of meaningless code words, or computer program—does not mean that none of the elements of the work can be copyrightable."). The same argument was used in Apple Computer, Inc. v. Microsoft Corp., 759 F. Supp. 1444 (N.D. Cal. 1991).

way as a machine part is made from a representation of the part in a technical drawing. Finally, a screen display is not an "article." It is a transient work in two dimensions. It is not like a statuette, belt buckle, or bike rack. Analyses of computer programs under the useful article doctrine give far too broad a reading to a narrow, limited rule.

2. Merger

In the United States, the courts have formulated the mysterious merger doctrine. When courts examine computer programs, they often confuse this doctrine with the idea-expression distinction, the rule against the copyright of systems, and the copyright of minimalist works.[131] The logic is as follows.

If the same idea can be expressed in a plurality of different manners, a plurality of copyright may result.[132] However, copyright protection will not be given to a form of expression necessarily dictated by the underlying subject matter.[133] If the idea can only be expressed in one way, then what is being expressed is not expression but an idea, and is not the proper subject matter of copyright. It is said that the expression and idea have merged where there are few or no ways of expressing a particular idea.[134] One reason for limiting the scope of copyright where there are few forms of expressing an idea (assuming it is possible to determine that the different expression is of the same idea) is to stop one person from copyrighting those few forms of expression by reducing to writing all possible forms of expression and taking the idea out of the reach of the public.[135] However, this assumes that the second person wishing also to express that idea has access to all the copyrighted expressions: If the second person independently comes up with the same expression, which is likely if the means of expression are limited as the doctrine supposes, then there has been no copyright infringement.

For computer programs, as shown in the previous section on patents,

131. *See, e.g.*, Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1253 (3d Cir. 1983).

132. *See* Dymow v. Bolton, 11 F.2d 690, 691 (2d Cir. 1926).

133. *See, e.g.*, Freedman v. Grolier Enters., Inc., 179 U.S.P.Q. (BNA) 476, 478 (S.D.N.Y. 1973).

134. *Apple Computer*, 714 F.2d at 1253, *referring to* Morrissey v. Procter & Gamble Co., 379 F.2d 675, 678–79 (1st Cir. 1967). *See also Lotus Dev. Corp.*, 740 F. Supp. at 58–59 ("When a particular expression goes no farther than the obvious, it is inseparable from the idea itself. . . . It is only a slight extension of the idea of 'obviousness'—and one supported by precedent—to reach the . . . concept 'merger.' If a particular expression is one of a quite limited number of possible ways of expressing an idea, then, . . . the expression is not copyrightable.").

135. *Morrissey*, 379 F.2d at 678–79.

the constraints of a programming language do not allow much room for variations in expression. Therefore, courts have looked at the computer program in issue, tried to determine its idea, and then examined whether that idea could be expressed in other ways.[136]

This test has problems. Courts in practice equate "idea" with "algorithm." But, as previously seen, the algorithm can be expressed at different levels of abstraction. There is no one algorithm for any program. Different programmers may state the algorithm of a program at many levels of abstraction and all be correct. For every program, its "idea" can always be expressed in many ways. Examples of differing expressions of an algorithm may include computer code, machine code, high-level pseudo code, and complex or simple flow charts.

Even if one only looked at computer code, the answer depends on what level of abstraction of the underlying idea is chosen. One can not say that the expression used is the only way of expressing the idea of the program. It may be the only way of expressing an idea as a program, but if one looked at the idea from another level, it could be expressed in other ways.

The test, if it is ever workable, encourages complex programs. All computer programs can be expressed in more than one way. To be safe, a programmer will express a simple algorithm in a roundabout way to show that there is more than one form of expression and that the form of expression chosen is more than an expression of the idea.[137]

The merger doctrine has expanded to catch cases where there are few forms of expression of the idea. But how does one determine how many is a few? If there is more than one way of expressing an idea, the court must decide which are really expressions of the idea at issue and which are expressions of a second idea. This is the same as determining if two

---

136. *See, e.g., Apple Computer*, 714 F.2d at 1253 ("The idea of one of the operating system programs is, for example, how to translate source code into object code. If other methods of expressing that idea are not foreclosed as a practical matter, then there is no merger."); Digital Communications Assocs. v. Softklone Distrib. Corp., 659 F. Supp. 449, 458 (N.D. Ga. 1987) ("Since the work at issue is the status screen, the court must determine what is the 'idea' behind the status screen and then determine whether the expression of the status screen is 'necessary' to that 'idea'.... Thus, 'idea' is the process or manner by which the status screen, like the car, operates and the 'expression' is the method by which the idea is communicated to the user.).

Taking from a program those elements that are essential to its purpose is regarded as taking of an idea and not infringement. *See also* M. Kramer Mfg. Co. v. Andrews, 783 F.2d 421, 436 (4th Cir. 1986) (Normally all computer programs will be expressions not ideas.).

137. Under the merger doctrine, would directions to run the Boston Marathon be copyrightable, even though there was only one path to run? What if the directions included nonessential information, such as good places to rest?

forms of expression are substantially similar. In other words, the merger doctrine states a conclusion. It is not a test.

To make matters worse, it is often alleged in cases in this area that the plaintiff is trying to copyright a system; that is to say, the plaintiff is trying to protect a procedure to carry out a particular task by claiming copyright over the instructions for the task[138] or over a form to be completed while accomplishing the task.[139] It is clear in copyright doctrine that it is only expression, and not a method or a system, that is the subject matter of copyright.[140] Even so, the reasoning in such cases is often that what is claimed as copyrightable is "so straightforward and simple" that "copyright does not extend to the subject matter at all."[141] In other words, the forms or instructions themselves are not the proper subject matter of copyright in such cases, for protecting them is tantamount to protecting the system since there are few ways of expressing that system.[142]

Baker v. Selden[143] is the foundation case in this area. Its holding is confused. The plaintiff, by copyrighting a book containing bookkeeping forms and instructions [illegible] those forms, claimed copyright in the method of bookkeeping. The Supreme Court, reasoning that "[t]he copyright of a work on mathematical science cannot give to the author an exclusive right to the methods of operation which he propounds, or to the diagrams which he employs to explain them, so as to prevent an engineer from using them whenever the occasion requires," concluded that "no one has a right to print or publish [the plaintiff's] book, or any material part thereof . . . [but] any person may practice and use the art

138. *See, e.g., Morrissey*, 379 F.2d 675, where the plaintiff claimed, unsuccessfully, copyright over instructions to enter a competition.

139. *See* Bibbero Sys. v. Colwell Sys., 731 F. Supp. 403 (N.D. Cal. 1988) (a blank form to determine procedures and diagnoses to be performed by doctors is not copyrightable), *aff'd*, 893 F.2d 1104 (9th Cir. 1990). *Cf.* Applied Innovations, Inc. v. Regents of the Univ. of Minnesota, 876 F.2d 626 (8th Cir. 1989) (computer software infringed copyright in test consisting of short simple statements; test data expression of process or facts).

140. In the 1976 U.S. Copyright Act, 17 U.S.C. § 102 (1988), this is made clear by § 102(b) which stops copyright extending "to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described."

141. *Morrissey*, 379 F.2d at 679.

142. The court in *Morrissey* rejected the reasoning that as "the substance was relatively simple, it must follow that the plaintiff's [instructions] sprung directly from the substance and 'contains no original creative authorship,'" but held that as the "subject matter is very narrow, so that 'the topic necessarily requires' . . . only a limited number [of forms of expression] . . . the subject matter would be appropriated by permitting copyrighting of its expression." Both tests lead, as a practical matter, to the same result. *Id.*

143. 101 U.S. 99 (1879).

itself which [the plaintiff] has described and illustrated" in the book.[144] However, the court ruled that "blank account-books are not the subject of copyright; and that mere copyright of [the plaintiff's] book did not confer upon him the exclusive right to make and use account-books . . . illustrated in said book."[145] This is the foundation of the so-called rule that blank forms are not copyrightable subject matter. Even literal copying, in this court's view, would not be a breach of copyright.[146] The court distinguishes the text of the book (which was copyrightable) from the forms shown in the book (which were not copyrightable). Nothing is said in *Baker* about the case where only one way to express the system or to design the forms exists, or where the expression merged into the idea.[147]

The Code of Federal Regulations Section 202.1 prohibits copyright of blank forms "which are designed for recording information and do not in themselves convey information," names, titles, slogans, and lists of ingredients and contents.[148] This regulation is said to be founded on the rule of *Baker v. Selden*.[149] But it is not correct to say, as some courts have held, that blank forms and the like cannot be copyrighted—the correct rule is that blank forms may be copyrighted "if they are sufficiently innovative that their arrangement of information is itself informative."[150]

Baker v. Selden has been expanded to computer software cases to claim, since the screen display resembles a blank form, that the screen display is an unprotected idea. This raises the issue whether giving

144. *Id.* at 103–04.

145. *Id.* at 107.

146. Would the result be the same today? Part of the court's reasoning was that in 1859 the copyright legislation then in force only gave copyright over "books, maps, charts, musical compositions, prints, and engravings." Today, the 1976 Act gives protection to "graphic" works. If blank forms conveyed information, thus not being useful articles, they could be regarded as graphic works, and the forms themselves would be copyrightable. However, if a form were not regarded as an "article" but only as a "work," then it would not even need to convey information to be copyrightable. This latter interpretation seems the most sensible.

147. *Cf.* Educational Testing Servs. v. Katzman, 793 F.2d 533, 539 (3d Cir. 1986) ("It is on the basis of the merger principle that copyright has been denied to utilitarian ideas, such as forms.").

148. 37 C.F.R. § 202.1(c) (1991).

149. *See, e.g.*, Bibbero Sys. v. Colwell Sys., 731 F. Supp. 403, 404 (N.D. Cal. 1988), *aff'd*, 893 F.2d 1104 (9th Cir. 1990). The regulation also prohibits copyright for tables containing public information, calendars, and tape measures, as they contain no original authorship.

150. Digital Communications Assocs. v. Softklone Distrib. Corp., 659 F. Supp. 449, 461 (N.D. Ga. 1987) (quoting Whelan Assocs. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1243 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987)).

copyright to the program in effect impermissibly protects a blank form. These claims misinterpret *Baker v. Selden*. The screen always gives information to the user. The only valid test, relying on this line of cases, should be whether the screen display contains enough expression to amount to an original work. Secondly, relying on *Baker*, because the computer program carries out a process, it has been suggested that the programmer should not be given copyright over the expression, as doing so would give the author the exclusive right over the process.[151] The validity of this argument will be examined below.

The merger doctrine has not reached Australia. When the subject matter of an action allows little variation in the form of expression, Australian courts usually decide that copyright subsists in the work. However, a precise similarity must be shown to exist between the two works before there will be a finding of infringement.[152] As an example, the design of a simple house is likely to have standard-height ceilings and an entrance hall adjacent to the front door. Copyright infringement will be found in Australia only if the plan of those features is copied exactly.[153] The merger doctrine in United States law, by contrast, does not give any copyright protection, even against literal copying, in circumstances where there are few ways of expressing the idea. In Australia, it is clear that copyright protection is given for the particular expression used by the author explaining a method of operating a system or using an apparatus or playing a game. The courts have not regarded this as giving copyright over the process carried out. It is irrelevent, in regard to the process, system, or game, whether the resulting expression is obvious.[154] Thus, not all arguments used in the United States will be available in Australia when dealing with computer software copyright.[155]

The merger doctrine should be abandoned by the courts for all copyright cases.

---

151. *See, e.g.*, Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1250–52 (3d. Cir. 1983). *Cf.* Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37, 54–58 (D. Mass. 1990).

152. *Compare* Dorsey v. Old Surety Life Ins. Co., 98 F.2d 872, 874 (10th Cir. 1938) ("a showing of appropriation in the exact form or substantially so") *with* Continental Casualty Co. v. Beardlsey, 253 F.2d 702, 705 (2d Cir. 1958) ("a stiff standard for proof of infringement").

153. *See, e.g.*, Dixon Invs. v. Hall, unreported, Federal Court of Australia (Pincus, J.).

154. *See, e.g.*, Meccano Ltd. v. Anthony Horden & Sons Ltd., 18 N.S.W. St. R. 606 (1918).

155. As the law currently stands, one would feel it would be harder to protect the nonliteral elements of computer programs in the United States than in Australia. But this has not been the case. The courts in the United States have decided, mostly on what seem like policy grounds, to give wide protection to computer programs and other aspects associated with the use of computers, such as interfaces.

C. The AutoCAD Example

No court has yet found infringement where neither the underlying algorithm of the program nor the user interface had been copied. The nearest courts have come is in the Australian *Autodesk* case.[156] The plaintiff, an American software company, wrote a complex Computer Aided Design ("CAD") program, AutoCAD. Part of this program contained another program, called "Widget C," which regularly sent a signal to a hardware device (called a lock, but really a key) attached to the computer's serial port. Unless the lock was attached, no display would appear on the screen and no printout of any drawings or calculations could be made.[157] The signal sent to the lock by Widget C was a series of electrical impulses, represented in binary by ones and zeros, which the lock processed. The lock then replied to the Widget C program: If the reply was the one expected, the CAD program continued to run as normal. The lock was a hardware device which included as components a clock, a shift register, an XOR gate and switching circuitry. The lock stored no computer program in the sense commonly understood by the computer industry.[158] Although the CAD (and therefore the Widget C) program was easy to duplicate, the lock was not.[159]

The object of the lock was to prevent the CAD program from running correctly if illegally copied. Copies run without the lock would not function. The lock did this by receiving a binary input and returning a binary output.

The defendant produced a software version of the lock (the "Auto-Key lock") which had the same function as the plaintiff's lock. However, the algorithm used by the defendant's lock to reply to the Widget C signal was completely different from the hardware components in the plaintiff's lock. The defendant determined how the lock worked by examining the output of the plaintiff's lock with an oscilloscope; consequently the defendant had no need to examine the Widget C program or

---

156. Dyason v. Autodesk Inc., 96 A.L.R. 57 (1990) (Full Federal Ct. Decision). Oral argument in the appeal took five days.

157. *Id.* at 68. Thus the lock prevented unauthorized use of the AutoCAD program that retails for A$5,700. Autodesk permits adaptation of its program under license and has done so approximately 100 times in Australia. *See* Autodesk's submission to ACLR, Collection, *supra* note 57.

158. *Id.* at 93.

159. *Cf.* Vault Corp. v. Quaid Software Ltd., 847 F.2d 255 (5th Cir. 1988), *commented on in* Deborah Kemp, *Limitations upon the Software Producer's Rights: Vault Corp. v. Quaid Software Ltd.*, 16 RUTGERS COMPUTER & TECH. L. J. 85 (1990) (Software lock that unlocked plaintiff's protection program did not infringe copyright in the protection program.).

the components making up the plaintiff's lock. The devices had the same function but used different algorithms to achieve their results. They had the same interface and they produced the same output given the same input.

The trial judge found there was copyright infringement[160] because: Each lock contained a computer program, the function of the plaintiff's program had been reproduced in a material form by the defendant, and the form of the reproduction was irrelevant.

The Full Federal Court reversed. Its findings were: (1) Copyright in computer programs resides in the expression of a set of instructions that cause a specified device to perform a particular function; but copyright does not reside in the function of the device or program itself.[161] (2) The lock, looked at in isolation, was neither a computer program nor a device which contained a computer program.[162] The information—the voltages—sent by the Widget C program to the lock was not an instruction to stop or proceed. The Widget C program decided whether to stop or proceed from the information returned from the lock. (3) The Widget C program and the defendant's lock were an integrated system, which constituted a computer program.[163] (4) The defendant did not reproduce any part of the Widget C program.[164] (5) The defendant's lock, whether it was a program or not,[165] was not a reproduction or adaptation of the plaintiff's Widget C program.[166] There was no reproduction because the only similarity was in the function of the locks, not the defendant's lock and the plaintiff's program.[167] (6) As the algorithms were different, the defendant's lock was not an adaptation or variant of the combination of

160. Dyason v. Autodesk Inc., 15 I.P.R. 1, 27 (1989).

161. *Dyason*, 96 A.L.R. at 65.

162. *Id.* at 78, 105.

163. *Id.* at 78.

164. "[T]he expression of the set of instructions being Widget C was not reproduced in the hardware lock. . . . [T]here was only one relevant computer program in the present case. This was the AutoCAD program itself." *Id.* at 104.

165. Sheppard, J., was responding to the plaintiff's submission that "Widget C was itself the program but a substantial part of that program was reproduced in the lock. Thus the computer had the program Widget C in it and the lock had part of Widget C in it. In other words Widget C was a program and each of the locks was a reproduction of that part of the program so that the comparison was Widget C and each of the locks." *Id.* at 76. However, if the lock had part of Widget C in it, it would contain a computer program, which the court found was not the case. The submission assumes that a program can be infringed by a piece of hardware replicating the function of part of the program.

166. The court relied on Australian Copyright Act § 14, which allows the court to see if the defendant's work infringed a substantial part of the plaintiff's work, in this case the Widget C program. *Id.* at 78–83, 105–06.

167. "[W]hat is contained in the Auto-Key lock is not a substantial part of the program because, in substance and in form, it is essentially different." *Id.* at 83 (Sheppard, J.)

the plaintiff's lock and Widget C program.[168]

The result of the case may be correct—the defendant copied function not expression. The defendant "invented" his algorithm independently and did not copy the plaintiff's algorithm, and, therefore, any computer program from the plaintiff. However, some of the reasoning of the court is suspect.

1. Can Hardware Be Software?

First, is the court's reasoning for not categorizing the lock as containing a computer program[169] correct under Australian law? By definition, an object can not be both a computer program and a computer.[170] If a computer program can be reproduced from any form of storage, whether the program is visible or not while stored, then what is stored, or more correctly the form of storage, is a reproduction or an adaptation of the computer program reproduced. Therefore, if the algorithm that causes the lock to convert the input into the output is stored in the lock and can be "reproduced" from that storage, then the lock contains a reproduction of a computer program. It is not correct, as the court did, to look at the reason the lock existed (to stop the effective operation of the AutoCAD program if not attached to the computer) and conclude because the lock did not do this directly, but just produced automatic responses, that it was not a program. The lock's function was to take binary input and

---

168. *Id.*

169. "'[C]omputer program' means an expression, in any language, code or notation, of a set of instructions . . . intended, either directly or after either or both of the following: (a) conversion to another language, code or notation; (b) reproduction in a different material form, to cause a device having digital information processing capabilities to perform a particular function." Australian Copyright Act § 10.

170. The Australian Copyright Act requires that a program cause a device (here a computer) to perform a particular function. This prevents a hardware device from being protected under copyright as a computer program. However, according to the Australian Copyright Act, a hardware device can "contain" a computer program, stored in hardware, if the program can be reproduced from the hardware. The form of storage containing the program is a "reproduction in a different material form" of the reproduced expression.

Charles Babbage, who died in 1871, designed a "difference machine" that could generate mathematical tables of many kinds. It operated by thousands of swirling intricate geared cylinders interlocking in incredibly complex ways. The machine used an algorithm, the "method of differences," which was physically built into the machine. Such an algorithm, unless it could be "reproduced" from the machine, would not be a computer program under the Australian Copyright Act. Compare Babbage's "analytical engine" that stored numbers and made decisions under the control of a program contained in punch cards. *See* DOUGLAS R. HOFSTADTER, GODEL, ESCHER, BACH: AN ETERNAL GOLDEN BRAID 25 (1979). *See also* Computer Edge Pty. Ltd. v. Apple Computer, Inc., 161 C.L.R. 171, 194 (1986). Hardware and software are logically equivalent. ANDREW S. TANENBAUM, STRUCTURED COMPUTER ORGANIZATION 11 (2d ed. 1984).

produce binary output.[171] This function may be carried out by a computer program[172] or by a piece of hardware, which although using an algorithm does not contain a reproduction of a computer program.[173] Determining the function of a device does not determine what it is.

The court instead determined that the lock and the Widget C program together constituted a computer program. But if the lock was not a computer program and did not contain a computer program itself, as the court determined, how can the lock, in combination with a computer program become a computer program? It is like saying that a piece of hardware, such as a computer keyboard, which operates in conjunction with a computer program, is part of the program.

The correct view is that the plaintiff's lock is just a piece of hardware. It is not a program and does not contain a program. It can not reproduce a program from any storage: It contains no form of storage apart from the shift register. It does, through its circuitry, follow an algorithm, just as a washing machine follows an algorithm, but this algorithm is not, in Australian copyright law, a literary work in a material form.

It should be the case, both in Australia and the United States, that the function a machine performs is not protected by copyright. That is the domain of patent. It is sensible that copyright does not protect function. To do so would give too a wide a scope to copyright. Preventing copying of functions would prevent anyone from creating a road map, recipe or spreadsheet: works whose functions have already been invented. Function does not involve expression. However, if a machine can reproduce a form of expression that causes a machine to perform a function, in the narrow sense that it can store and then reproduce (print out) the instructions that cause the machine to operate, that expression should be

171. In this case, the binary code transmitted was not a set of instructions (to stop), but input or output data, which Widget C acted on. There is nothing to stop a hardware device from transmitting to another hardware device a computer program (a set of instructions) in the form of binary code, such as a program sent via a modem to a computer from another computer, which causes the receiving computer to act in a certain way.

172. The lock could be attached to the printer port. A printer receives instructions from a computer program, called the operating system, to print certain data. The printer tells the program, by binary code, when it has finished this task. How is this different from the lock in this case? Many printers have a separate computer program installed in them in ROM to enable the printer to communicate with the operating system. In Star Micronics Pty. Ltd. v. FiveStar Computers Pty. Ltd., unreported, Federal Court of Australia, Oct. 9, 1990 (Davis, J.) (holding that computer program embedded in computer chip in printer was protected by copyright).

173. The algorithm could be expressed, for example, in pseudo-code. *See* Dyason v. Autodesk Inc., 96 A.L.R. 57, 75 (1990) (Full Federal Ct. Decision).

copyrightable.[174] Otherwise computer programs would be deprived of protection in their most useful form.

2. *Can Hardware Be a Copy of Software?*

Secondly, assume arguendo that the plaintiff's lock contains a computer program. If the defendant's lock does not contain a computer program, could the defendant's hardware infringe a copyright on the plaintiff's program? The Full Federal Court, in asking the question whether the defendant's lock was a reproduction of a substantial part of the Autodesk program, assumed that a piece of hardware may, under some circumstances, infringe the copyright of a computer program. The court indicated that infringement occurs when there is similarity in the sets of instructions constituting the programs.[175] But how can there be such similarity when one object being compared does not contain a set of instructions. True, the defendant's lock operates in accordance with instructions that could be written as an algorithm, but the expression of that algorithm is not determined by the workings of the lock. A piece of hardware, not containing a computer program, can never be a reproduction of a computer program, whether that computer program is written down or is stored and can be retrieved from another piece of hardware.

The Full Federal Court then examined whether the defendant's lock, a piece of hardware not containing a computer program, was an adaptation of the plaintiff's Widget C program.[176] The Australian Copyright Act defines "adaptation" in this context as "a version of the work (whether or not in the language, code or notation in which the work was originally expressed) not being a reproduction of the work."[177] It is implied in this definition, and the definition of "material form," that the work in question must be a form of expression. The forms of expression are examined to determine whether there has been a copying of expression—to see if one form of expression is a version or variant of the other form of

174. *But see* Davies & Co. v. Comitti, 54 L.J. Ch. 419 (1885); STANIFORTH RICKETSON, THE LAW OF INTELLECTUAL PROPERTY ¶ 5.34 (1984).

175. *Dyason*, 96 A.L.R. at 66, 82, 105.

176. *Cf.* United States definition of "derivative work": "a work based upon one or more pre-existing works, such as a translation, musical arrangement, dramatization, . . . or any other form in which a work may be recast, transformed, or adapted." 17 U.S.C. § 101 (1988).

177. Australian Copyright Act § 9.

expression. As the lock expresses nothing, it could never be an adaptation of a computer program.[178]

It is also incorrect to examine whether the lock is a three-dimensional reproduction of a two-dimensional computer program or algorithm.[179] Computer programs and algorithms are not artistic works. An analogy can not be made to the Australian house plan cases[180]—although both a house plan and an algorithm tell a "processor" how to accomplish a task (how to build a house or respond to binary input). The house results from following the algorithm and is an artistic work itself protected by copyright. The lock does not result from carrying out the algorithm; rather it performs the algorithm. It would be ridiculous to say that a house executes a house plan.

The law should be clearer: A machine that performs the same function as a computer program should never be a copy, reproduction, adaptation, or derivative of the computer program. A machine that uses the same algorithm as expressed in a computer program can never be a copy of the computer program. In both cases, there is no copying of expression.

The current copyright laws in Australia and the United States raise several concerns over their applicibility to the growing field of computer programming. Below is a proposed test that would correct the problems inherent in the two countries' systems. The test's application to the AutoCAD case will illustrate its advantages.

D. When Are Two Computer Programs the Same? A Proposed Test

If the defendant's lock did contain a computer program, then the court would have to decide if there was substantial similarity between the plaintiff's program and the defendant's program. The Australian court was correct in holding that functional similarity is not sufficient for copyright infringement: Copyright protects expression, not function.[181]

---

178. *Cf.* Computer Edge Pty. Ltd. v. Apple Computer, Inc., 161 C.L.R. 171, 186 (1986) ("An adaptation must itself be a 'work.'").

179. An artistic work in Australia is deemed to have been reproduced, in the case of a two-dimensional work, if a version of it is produced in a three-dimensional form, or, in the case of a three-dimensional work, if a version of it is produced in a two-dimensional form. Australian Copyright Act § 21(3).

180. *See, e.g.*, Collier Constrs. Pty. Ltd. v. Foskett Pty. Ltd., 97 A.L.R. 460 (1991); Dixon Invs. v. Hall, 18 I.P.R. 490 (1990). *See also* Hart v. Edwards Hot Water Sys., 159 C.L.R. 466 (1985).

181. But note the court went too far in Vault Corp. v. Quaid Software Ltd., 847 F.2d 255, 268 (5th Cir. 1988), holding that if two programs had the same code, but different functions, they would not be substantially similar.

But what if the two programs, in addition to carrying out the same function, use the same algorithm, although expressed in different ways? If different computer languages were used, and the court was able to determine that what was expressed was the same algorithm, then one work would be a reproduction or adaptation of the other. For example, a program in Pascal and a compiled version of that program are the same program, in the same way that *Crime and Punishment* is the same novel whether in Russian or English (even though there may be variations of expression between two English translations). It is a similar analysis where the Pascal program is translated, line by line or construct by construct, into another language at the same level of abstraction. But what if, like the defendant in the *Autodesk* case, the defendant never saw the plaintiff's expression, but rather "guessed" the algorithm used and wrote a program in a different language that used that algorithm. In other words, should copyright protect an algorithm expressed in the form of a computer program and prevent others from expressing that algorithm in another way?

To restate the problem, copyright protects more than the literal expression, but does not protect ideas[182] or function. Two computer programs may use the same method to accomplish the same goal. That method is not protected.[183] Copyright does not protect an algorithm, but only expression of the algorithm. Although not protected by copyright, a program's algorithm must be examined to determine if there is copying of expression. If algorithms are ignored and function is not examined, there is no way to determine if two programs, expressed in different languages or at different levels of abstraction, are similar: A computer programmer will usually look to see if, taking the same input, the same output will result (functional similarity), or if the algorithm that each program uses is the same.[184] As copyright ignores function, the test for substantial similarity between two programs written in different languages must be, at a minimum, whether the underlying algorithms are the same.

No test should give copyright protection to algorithms.[185] An

182. *See* Ashton-Tate Corp. v. Ross, 916 F.2d 516, 521 (9th Cir. 1990) (to be author of spreadsheet, one must contribute more than the idea).

183. *See* 17 U.S.C. § 102(b) (1988). *Cf.* Brigid Foley Ltd. v. Elliot, [1982] R.P.C. 433, 434.

184. There are many programs that translate from one language to another. The program translated may just be one of a number of possible translations; a program could be copied from another program but when the original program is translated, a different version is likely to result. In each case the algorithm will be the same.

185. Algorithms, being equated with methods or processes, are not expressions and are not copyrightable. Compare the Japanese position that does not give copyright protection to "methods of solution" ("kaihoo"), which Japanese courts have interpreted as including algorithms. *See* Dennis S. Karjala, *Japanese Courts Interpret the "Algorithm" Limitation on the Copyright Protection of Computer Programs*, 31 JURIMETRICS J. 233 (1991).

alternative test to those currently used must meet this goal. A computer program should be protected as expression. The algorithm, if written down in the form of a flow chart or pseudo-code, does not cause a computer to perform any particular function, and should not be copyrightable as a computer program.[186] The algorithm, expressed in this form, may be copyrightable as an artistic or literary work in its own right, just as a house plan or a recipe may be copyrightable. This copyright does not prevent the builder or chef from using the plan or recipe.[187] However, for this algorithm to be used by a computer, it must be "translated" by the programmer into another form of expression, the computer program, and one might argue that the program (as a translation) infringes the copyright in the flow chart[188] or pseudo-code. As a result, any programmer who uses the flow chart or pseudo-code to write a program infringes the copyright in the flow chart or pseudo-code.[189] Further, if a second programmer deduces the algorithm from the original program, and uses it to write a program, there is indirect reproduction of the first programmer's flow charts and pseudo-code.[190] Therefore one may decide that in effect the algorithm is copyrightable.

To prevent this, but to allow courts to look at the underlying algorithm to determine if two programs are substantially similar, computer programs must be partitioned from other works in the copyright regime. Computer programs should not be regarded as literary works.[191] The copyright of a literary work, artistic work, or any other work, should not be infringed by a "computer-program work" and vice versa.[192] A computer program is a work whose intention is to cause a computer to perform a certain task. Therefore, a novel stored on a computer disk is not a computer program, as it does not cause a computer to do anything. An algorithm expressed in the form of a diagram is copyrightable as an

186. However, in Australia, such expression of the algorithm is copyrightable as a computer program. *See* Australian Copyright Act § 10 (definition of "computer program").

187. *See* Cuisenaire v. Reed [1963] V.R. 619, 736.

188. "Flowcharts . . . are works of authorship in which copyright subsists, provided they are the product of sufficient intellectual labor to surpass the 'insufficient labor hurdle'." CONTU FINAL REPORT 43 (1978), *cited with approval in* Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37 (D. Mass. 1990).

189. *Cf.* Synercom Technology, Inc. v. University Computing Co., 462 F. Supp. 1003, 1013 n.5 (N.D. Tex. 1978) (coding "detailed description of particular problem solution, such as flowchart" was violation of copyright); Data Cash Sys. Inc. v. JS&A Group Inc., 480 F. Supp. 1063, 1067 n.4 (N.D. Ill. 1979).

190. *Cf.* Solar Thompson Eng'g Co. Ltd. v. Barton, [1977] R.P.C. 537; Purefoy Eng'g Co. Ltd. v. Sykes, Boxall & Co. Ltd., 72 R.P.C. 89 (1955).

191. It is interesting to note that Autodesk, in its submissions to ACLR. Collection, *supra* note 57, at 3, states that "computer programs should not be treated as literary works."

192. In other words, a non-computer program can not be a derivative work of a computer-program work. *But see* Williams v. Arndt, 626 F. Supp. 571 (D. Mass. 1985).

artistic work, but a program written using that algorithm does not infringe the copyright in the diagram. A piece of hardware that uses an algorithm to carry out its task, but does not allow the storage and retrieval of the algorithm, does not infringe the copyright in a computer program that uses that algorithm.[193] Only computer programs can infringe the copyright in computer programs.[194]

The reason for this limitation is the idea-expression distinction. To determine if there is copying, one must compare expression. If the rule allowed comparison between an algorithm expressed as a computer program and an algorithm expressed as a diagram, it would be too much like a comparison of ideas. As the expressions and levels of expression would be substantially different in virtually every case, there can be no infringement.

Computer programs are different when they use different algorithms. But they are not always similar when they use the same algorithm. A test of substantial similarity of expression must do more than determine whether the underlying algorithms of two programs are the same.

The same algorithm can be expressed at different levels of abstraction, from a general high-level description of the method of completing the task, to a detailed low-level description. Or looking at it another way, each program's algorithm can be expressed at different levels of abstraction, from the high level (read in data, process data, or print monthly report) to the low level (a computer program written in assembler language.) Again, the idea-expression distinction becomes useful. So far most courts in the United States have used this dichotomy, and, unfortunately, the merger doctrine, to determine if what is expressed is essential to the program's function. Alternatively, if there are various means of expressing the function of the program, then what is chosen by the author as expression is protected expression. But there is always more than one way of expressing the function of a program.

What the courts should concentrate on is the level of expression.[195] The algorithm of the program can be expressed at various levels, but not all those levels should be infringements of another program's expression.

---

193. *Cf.* Note, *Computer Intellectual Property and Conceptual Severance*, 103 HARV. L. REV. 1046, 1055 (1990) (Distinguishing hardware and software is meaningless.).

194. A narrower and unsatisfactory argument was used by the defendant in Computer Edge Pty. Ltd. v. Apple Computer Inc., 161 C.L.R. 171 (1986), that a reproduction must be in the same form or nature as the original form in which the alleged reproduction is made. That is, a reproduction of source code written on paper could only be infringed by the same source code written on paper. The proposed test in the text above says computer programs can only be infringed by similar computer programs.

195. *But see* Dennis S. Karjala, *Copyright, Computer Software, and the New Protectionism*, 28 JURIMETRICS J. 33, 87–92 (1987).

Clearly, the level of expression that the author uses to express the algorithm, which is in effect the literal code, is protected from copying. The translation of this code into a more detailed level of expression by a computer's compiler is a copy of the program: It is necessary to do so to run the program, and the low-level code results directly from the programmer's expression. The high-level description of the algorithm, which a computer cannot execute without the further efforts of a programmer, should not be regarded as a copy of the program. This expression is not a computer program, but an idea for a computer program. Any computer program that uses that high-level algorithm should not breach the copyright of another program that uses the same algorithm. For programs where the levels of expression differ but the algorithms used are the same, the test of substantial similarity should ask whether the differences in level of expression are such that the expressions themselves are different. This test is one of degree.

The test implicitly takes into account the idea-expression distinction. In difficult cases, the court should first determine the algorithm each program uses. If the algorithms are different, the inquiry should end there: There is no similarity of expression as what is being expressed is different. If the algorithms are the same, the court then decides at what level of abstraction they are the same. If it is at such a high level of abstraction that when the algorithm is expressed a computer could not execute the algorithm without the assistance of a programmer refining the level of abstraction, what the court is comparing for copying is not expression, but idea. Thus, there is no copyright infringement. If the algorithms are the same at a level of abstraction that a programmer can directly use to write the same program, without substantially changing the level of abstraction, then the two programs have expressed the same algorithm, although in different languages or styles. Thus, there is substantial similarity.

E. The Proposed Test in Action

Under such a regime, *Autodesk* is an easy case. There is no copyright infringement. The defendant's lock is hardware only, and cannot infringe the copyright of the plaintiff's computer program. Secondly, the algorithm the defendant used was different, so even if the locks each contained a computer program, they are not substantially similar so as to result in a finding of infringement.

What about *Whelan*?[196] There the court held that copyright

196. Whelan Assocs. v. Jaslow Dental Lab., Inc., 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987).

protection of computer programs may extend beyond a program's literal code to its structure, sequence, and organization.[197] A witness for the plaintiff testified that the file structures and screen outputs of the two programs were virtually identical and that five important subroutines within both programs performed almost identically in both programs. A witness for the defendant testified that there were substantive differences in programming structure, in algorithms, and in data structure, but that both programs had overall structural similarities.[198] The court regarded the programs as utilitarian works and decided to treat as irrelevant to its inquiry (as being idea) the purpose and function of the programs and everything necessary to that purpose or function.[199] But the evidence the court quotes[200] does not make it clear that the court did look at the structure of the programs rather than the function certain parts of the programs performed.[201] Further, the court did not examine the two works in whole, but only those parts identified as being similar by the plaintiff.[202] The Third Circuit decided that the programs were substantially similar.

The strongest case for the plaintiff is that the two programs used the same high-level algorithm for the programs' overall design, and the same algorithm in five parts of the programs. Even if these five subroutines were copied, that is not enough to show that the defendant breached the copyright in all of the plaintiff's program: It would only be so if those five parts were a substantial part of the work as a whole. Secondly, the evidence was that the basic structures of the two programs were the same, but that the algorithms differed substantively. What this may mean is that, at a very high level, the programs performed the same tasks in the same order and that the algorithms in their most abstract form were similar. That being the case, on the above analysis, there is no copyright infringement. The structure of a program (in this case another name for the high-level algorithm) must be refined before a program can be written, and so both programs having the same high level algorithm

197. *Id.* at 1237–38; *see* Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 STAN. L. REV. 1045, 1084 (1989) ("[T]he Whelan court naively reasoned that because a function could be performed in more than one way, its structure, sequence, and organization is expressive and therefore copyrightable.").

198. *See Whelan*, 797 F.2d at 1228.

199. *See id.* at 1235–38.

200. *See id.* at 1246–48 (the court highlighting the testimony "if we look at the functions done by the programs in order, we find that they are the same").

201. *See* Walter, *supra* note 38, at 132–33.

202. *Whelan*, 797 F.2d at 1245–46. *Cf.* Atari Games Corp. v. Oman, 888 F.2d 878, 882–83 (D.C. Cir. 1989) (rejecting component by component analysis and ruling that the court must focus on the "work as a whole").

but different expressions of that algorithm are not substantially similar. Therefore, there is no infringement.

The Eastern District of New York recently refused to follow the *Whelan* test for substantial similarity, calling it "inadequate and inaccurate."[203] The court instead applied what it called the "abstractions test," examining each level of generality of the alleged infringing computer program (the object code, the source code, and the "general outline") for substantial similarity with the same level of generality as the copyrighted program.[204] Where the program was found to be substantially similar at any level of generality, the court then examined that level to see if it was "important."[205] If not, the court decided that there was no substantial similarity.[206]

The court was correct to abandon the *Whelan* test. The reasoning of the court was, first, that a program could include more than one idea.[207] Thus, determining what was non-essential or unnecessary to that idea might be impossible. In addition, each program was made up of subprograms which had separate ideas and could be individually copyrighted. Thus, the court discredited the use of the merger doctrine in computer software cases. Secondly, the court divided "the structure of the program," a term used in the *Whelan* decision, into two components: the static structure (the structure of the program as text) and the dynamic structure (the order of execution of the program's instructions).[208] The court decided to examine only the static structure of the program—the dynamic structure being equivalent to a "process, system, [or] method of operation," which under section 102(b) does not receive copyright protection.[209]

In examining the static structure of the program, the court looked for substantial similarity at each level of generality. One version of the defendant's program directly copied thirty percent of the plaintiff's source code. Infringement was found.[210] With regard to a second version of the defendant's program, rewritten to avoid direct copying, the

203. Computer Assoc. Int'l, Inc. v. Altai, Inc., 1991 Copyright L. Rep. (CCH) ¶ 26,783, 24,611 (E.D.N.Y. Aug. 9, 1985).

204. *Id.* at 24,612 (citing Nichols v. Universal Pictures, 45 F.2d 119, 121 (2d Cir. 1930), *cert. denied*, 282 U.S. 902 (1931)).

205. *Id.* at 24,613.

206. *Id.* at 24,614.

207. *See id.* at 24,611.

208. *Id.* at 24,611–12.

209. *Id.* at 24,612.

210. *See id.*

court found no infringement.[211] When looking at the source code, the importance of the code copied, and not the number of lines copied, was examined. However, as the defendant had rewritten the code, there were no lines of code identical to those in plaintiff's program. The court therefore found no similarity at this level.[212] At the next level of generality, the interface with the operating system, the court found similarities, but held them to be "dictated by external factors" and not infringing.[213] Finally, the high-level structure was not substantially similar as "it was not important, because it was so simple and obvious to anyone exposed to the operation of the program."[214]

The court should not have limited itself to examining whether the lines of code were identical. A work may be substantially similar even though there is no literal copying of any particular element of the work. By ignoring the flow of control (the dynamic structure) of the program, the court ignored an important factor in determining the quality of the parts copied. It was as if the court, in determining whether a piece of music was copied, ignored the sound produced and only looked at the way the notes were arranged on the sheet music. When there is no literal copying, the flow of control, or more precisely, the algorithm of the program, must be considered. If it is the same, there may be copying of expression. As the algorithm was expressed at the same level in each program, the court should have determined, first, if it was the same algorithm being expressed, and second, if the expression was substantially similar, not if the expression was identical.

The court should not have considered whether the higher-level features copied were simple, obvious, or dictated by external factors while determining if what was copied was substantial. The words of a song may be simple, and a map may be dictated by external factors, but literal copying of those works is copyright infringement. When examining the high-level structure of the program (the high-level algorithm) the court would have been more correct if it found non-infringment on the ground that the high-level structure was an unprotected idea. That is, the court could have determined that since the algorithm required more refinement before a program could be written, the high-level structure was akin to idea, rather than expression.

---

211. *See id.* at 24,613–14.
212. *See id.*
213. *Id.* at 24,613.
214. *Id.*

IV. APPLICATION OF THE NEW TEST

A. User Interfaces

This Section uses the foregoing analysis and proposed test to address legal protection of user interfaces. The first Subsection defines user interfaces and distinguishes between interface specifications and interface implementations. Succeeding Subsections describe an example of a copyright dispute concerning a user interface and explore various rationales for providing user interfaces with copyright protection. The final Subsection applies the proposed test and concludes that only user interface implementations should be accorded copyright protection.

1. User Interfaces Defined

A user interface is a set of rules or conventions allowing a human to communicate with a computer program. In analyzing user interfaces, it is important to distinguish between interface specifications and interface implementations. To illustrate the distinction, consider a user working with a word processing or spreadsheet program.

The interface specification is a set of abstract rules that might be implemented in any number of ways. For example, a rule that pressing the F1 key causes the word processor or spreadsheet to save a file to disk is an element of an interface specification. This rule is general; it says nothing concerning how the user's pressing the F1 key is processed by the word processor or spreadsheet. An interface implementation, on the other hand, is the computer code that translates the interface specification rules into action. The interface implementation is itself a computer program that stands between the human user and the word processor or spreadsheet.[215] When the user presses the F1 key, the interface implementation translates that action *in a particular way* into commands that cause the word processor or spreadsheet to save the file to disk.[216]

---

215. Besides user interfaces, there are several other types of interface programs which act as intermediaries between different components of a computer system. For example, communications interfaces allow remotely situated computer systems to communicate by setting a common communications protocol. *See* Fujitsu Australia Ltd., submission to ACLR, Collection, *supra* note 57, at 127.

216. The distinction between interface specification and interface implementation can be central in resolving disputes between parties concerning rights to user interfaces. One example is the IBM-Fujitsu operating system dispute. The dispute was resolved by arbitration on November 29, 1988, allowing Fujitsu to derive specifically defined interface information from new IBM programming materials, in return for payment of an annual access fee. Fujitsu was given access only to interface information (to allow it to design application programs for the IBM operating system environment) that describes the program's function, not its implementation. The specifications shared "do not describe the Program's

As the interface implementation will work to translate the user's pressing the F1 key into the "save to disk" command with either the word processor or the spreadsheet, it is independent of those programs.[217] The interface specification, which is the set of rules or principles underlying the interface, defines the operation of the interface implementation, which is the code used to implement the interface specification.[218] This Article concludes that only the implementation deserves copyright protection, the specification being non-protected idea.[219]

B. Legal Protection of User Interfaces: The Lotus *Case*

The extent to which the law should protect user interfaces has been highly controversial. This Subsection will focus on the copyright dispute concerning the user interface of the popular spreadsheet program "Lotus 1–2–3." In the *Lotus* case, the District Court of Massachusetts ruled that the defendants were liable for breach of copyright because they "copied protected nonliteral elements of expression in the user interface and the underlying computer program." According to the plaintiff, the user interface of the program included such elements as "the menus (and their structure and organization), long prompts, the screens on which they appear, the function key assignments, [and] the macro commands and language."[220] The court explicitly stated that it did not hold the defendants liable for copying the screen displays of the spreadsheet.

To resolve the issue of copyrightability, the court examined the work

---

structural or detailed design, internal component or module interfaces or other implementation details." Unpublished Arbitration Decision at 3. The decision also states if "an operating system's interfaces have been clearly defined, then relatively little information beyond that defined by one vendor as its products' customer interface specifications may be needed to independently develop a compatible operating system that allows customers to run existing application programs written for the original operating system." *Id.* at 11.

217. "For the most part, interfaces are defined at design levels higher than and independent of a product's implementation in detailed design or code." *Id.* at 12.

218. The rules contained in the interface specification convey no information about the contents of a particular interface implementation. For example, AT&T, SYSTEM V INTERFACE DEFINITION MANUAL (freely made available by AT&T) says "The System V Interface Definitions specifies an operating system environment that allows users to create application software that is independent of any particular hardware.... The functionality of components is defined, but the implementation is not."

219. *See, e.g.*, Wendy J. Gordon, *Merits of Copyright*, 41 STAN. L. REV. 1343, 1446–48 (1990).

220. Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37, 80 (D. Mass. 1990). Note that Lotus based its program on that of Visicalc, whose copyright was obtained by purchasing the corporation owning the Visicalc copyright. *See* SAPC, Inc. v. Lotus Dev. Corp., 921 F.2d 360, 361 (1st Cir. 1990).

to see where along the scale of abstraction of ideas the idea of the work fell. The court then determined whether the expression of that idea included elements of expression not essential to every expression of the idea, and if those elements were a substantial part of the work. The court did not use the "look and feel" concept to distinguish between non-literal elements of a computer program that are copyrightable and those that are not: "Look and feel" is a conclusion, the court said, not a test.[221] The court considered a number of ideas in the work, including ideas such as "an electronic spreadsheet," "a two line moving cursor," and the "designation of a particular key as a command key."[222] The ideas of an electronic spreadsheet, a structured menu, and a two line moving cursor may be expressed in numerous ways. These ideas were functional, obvious, and widely used. However, because the ideas may be expressed in a variety of ways, the court concluded that particular expressions of those ideas are copyrightable.[223] The designation of a "/" as a command key and the resemblance of the screen display to a paper spreadsheet, however, were present in most expressions and thus not "a copyrightable element of a computer program."[224]

The court did not look at the algorithm that the plaintiff's program used to implement these features. Instead, certain features of the outward appearance of the program when running were said to be original expression. As they were essential to a user's operation of the program, these features were copyrightable. Here the court treated the implementation of several of the features of the user interface as part of a literary work. Using this reasoning, a new type of lens for a movie projector, designed for a movie filmed in a particular way, is only one of the many lenses that could be used in a projector, and would be copyrightable because it is essential to the showing of a particular motion picture. The *Lotus* court was not concerned that the interface was not itself a literary work or independent computer program.[225]

Nor did the court decide if the literary work, the spreadsheet program itself, was copied. The plaintiff's program and the defendants' programs both had the same command tree and similar menu structures, and the court concluded that the defendants copied the expression embodied in

221. *Lotus*, 740 F. Supp. at 62–63.

222. *Id.* at 65–68.

223. *Id.* at 66–67. The court also said "That the defendants went to such trouble to copy [the user interface] is a testament to its substantiality," *Lotus*, 740 F. Supp. at 68.

224. *Id.* at 66.

225. *Cf.* Computer Edge Pty. Ltd. v. Apple Computer Inc., 161 C.L.R. 171, 214 (1986) (Executing the program's instructions does not reproduce or adapt the actual written program in which copyright subsists.).

the Lotus 1–2–3 menu hierarchy.[226] The court did not, however, analyze the program implementing the menu heirarchy or other features of the interface. As the *AutoCAD* case demonstrates, a similarity of function does not necessarily imply a similarity in implementation. Thus, the *Lotus* defendants may have used completely different algorithms and programs to implement the interface features of Lotus 1–2–3. Although the defendants used Lotus's interface specification, the court did not determine whether they copied Lotus's interface implementation.

C. Rationales for Legal Protection of User Interfaces

A complex user interface contains much original expression, takes many hours to develop, and is essential to the operation of a sophisticated computer system. This does not, however, lead to the conclusion that the copyright of the program producing and using that interface has been violated when its interface specification is copied.[227]

Since the *Lotus* decision, there has been much debate in the software industry as to whether user interfaces should be legally protected at all. Apart from where the computer code of the interface implementation has been directly copied, many software producers believe that a user interface is public property. This Subsection examines justifications for legally protecting user interface specifications.

1. Protection Needed to Promote Development

One argument for protecting interface specifications is that much work is put into designing such specifications and that consumers place a high value on good specifications in selecting programs. Much of the cost of creating a user interface is incurred in formulating the specification.[228] Only twenty percent of the cost of creating a computer

---

226. *Lotus*, 740 F. Supp. at 70. The menu hierarchy is one element of the Lotus 1–2–3 user interface.

227. A Lotus employee, referring to the Lotus 1–2–3 user interface, stated the work "is in the detail and the degree." The implementation of the interface, rather than its functionality, he said, should be protected. The *Lotus* court did not look at the program implementing the interface to see if it was copied. Frank Ingari, Forum at MIT on Intellectual Protection of Software (Oct. 30, 1990). At the same forum, the chief counsel of Lotus said Lotus only sued people who copied the whole interface.

228. *See* Lotus Dev. Corp., submission to ACLR, Collection, *supra* note 57 ("[I]t is widely recognized that the design of the user interface is a task which often requires greater creativity, originality and insight than the actual writing of the code. To deny copyright protection for the user interface would allow the misappropriation of those aspects of the computer program which entail the greatest investment in material and intellectual resources and which, in the case of Lotus 1–2–3, are the elements which have most contributed to its success."). *See also Lotus*, 740 F. Supp. at 68.

program (including the program's user interface component) is spent expressing the algorithm in the form of computer code.[229]

One response is that much effort is put into writing a history book. Indeed, it may take considerably more time and effort to do the research than actually to write the book. Nevertheless, it is only the book itself (the expression) and not the research effort that copyright protects.[230] Protection of a user interface presents a somewhat more difficult problem, however, because screen displays and other components of interface specifications give computer users information in a particular way about how to use a computer program. Because the information could be expressed in a different way, part of the interface specification is in fact an expression of a particular method of interaction between the program and the user.

2. *The Need for Incentive*

Another argument for legal protection of user interface specifications is that if such protection is not given, the incentive to develop new interfaces will decline sharply.[231] The plaintiffs in the *Lotus* case assert that "the tremendous growth and success of the U.S. software industry is the direct result of the creative and original efforts of its software developers, laboring under the protection of the copyright laws. Innovation has been the key to market success . . . ."[232] The plaintiffs also argued that the copyright laws protect "the lonely and defenseless developers working out of their dens and basements"[233] from having their work purloined by heartless corporations. Thus, the argument runs, if user interfaces are

---

229. WERNER L. FRANK, CRITICAL ISSUES IN SOFTWARE: A GUIDE TO SOFTWARE ECONOMICS, STRATEGY, AND PROFITABILITY 22 (1983).

230. *See, e.g.*, Miller v. Universal City Studios, Inc., 650 F.2d 1365 (5th Cir. 1981); Nash v. CBS, Inc., 899 F.2d 1537 (7th Cir. 1990); International News Serv. v. Associated Press, 248 U.S. 215 (1918); Nichols v. Universal Pictures Corp., 45 F.2d 119 (2d Cir. 1930). *Cf.* Jarrold v. Houlston, 69 Eng. Rep. 1294, 1298 (Chancery 1857).

231. *See generally* Australian Information Industry Association, submission to ACLR, Collection, *supra* note 57 ("Intellectual property primarily results from the application of human capital. . . . [C]opyright is related to improving market mechanisms by ensuring that owners or licensees of intellectual property achieve an adequate return on investment and effort. If protection were not provided, market mechanisms may not produce an adequate or desirable amount of intellectual property. . . . [T]he critical downstream impact on the economy of software as a production tool would be lost.").

232. Plaintiff's Post-Trial Brief at 75, *Lotus*, 740 F. Supp. 37. It adds, "The history of this industry has been one of creative designers who identify an unfilled need in the market and then design and build a superior product to fill that need. . . . [T]he developers' ability to realize substantial rewards for their creative efforts has depended entirely upon the legal protection copyright has afforded their work."

233. *Id.*

not protected, the biggest losers would be the small developers.

This argument is not entirely satisfactory, however, because whether or not copyright laws have encouraged software innovation generally,[234] they are far from the only incentive that software developers have for creating novel user interface specifications and implementations. Many improvements in interface design have been prompted not by copyright protection but instead by advances in hardware technology. The development of mouse-based graphic user interfaces, for example, depended on the availability of the high-resolution display screen.[235] Moreover, some computer users may value the interface specification more highly than the underlying algorithm of the program.[236] Therefore, competition among software developers to sell programs would provide incentive to create more attractive user interface specifications and implementations, independent of copyright law. Two competing software developers with equivalent programs would innovate interface specifications to gain a competitive edge.[237]

In response, an advocate of copyright protection for interface specifications might argue that without legal protection, competition among developers will not result. Consider the situation in which two developers each design programs performing the same function, but using different algorithms. Assume that each developer has used precisely the same amount of resources in developing its program. Suppose further that the second developer copies the first's interface specification (but nothing else) and thus incurs only the costs associated with integrating the interface specification into its own program. The first developer has expended resources innovating the interface specification, while the second is a free-rider. Having incurred no costs in developing a new interface specification, the second developer may now sell its product

---

234. *See, e.g.*, Gordon, *supra* note 219, at 1446–48.

235. Bill Curtis, *Engineering Computer "Look and Feel,"* 30 JURIMETRICS J. 51, 77 (1989).

236. This valuation does not itself provide a reliable indicator of the desirability of copyright protection. The mere fact that a consumer values a particular aspect of a computer program has not traditionally been used to determine the availability of legal protection. For example, consumers value program upgrades and clear reference manuals, which copyright laws protect. On the other hand, most consumers also value the accuracy of a program's results, its speed, and the reputation of its manufacturer, which copyright laws do not protect.

237. Bull HN Information Systems claims that having standard operating system interfaces increases competition, by allowing users to be able to choose computer elements from different suppliers and still be able to have them work as an integrated system, preventing the user from being tied involuntarily to one supplier. *See* submission to ACLR, Collection, *supra* note 57, at 4. A similar argument is that retraining costs involuntarily tie a user to one software-user interface, decreasing competition.

more cheaply, thereby gaining a price advantage in the market. The first developer would therefore rationally divert resources to other program features which are harder to copy or which are legally protected.[238] Therefore, without copyright protection of interface specifications, new programs will be produced, but they will contain no improvements in user interface design.[239]

This reasoning is superficially plausible, but it ignores several factors. First, although the costs of program design significantly exceed the costs of program implementation, it may be that the costs of interface design are only a small portion of the costs of developing a new program.[240] Thus, any price advantage gained by a developer copying another's interface specification may be small. This advantage would be reduced or even eliminated if a developer copying an interface specification incurs greater costs in creating a corresponding interface implementation than the creator does.

The creator of the specification also obtains lead-time advantages. By being the first to market, the creator will, for some period, enjoy a monopoly on sales of the new user interface. Moreover, the creator indirectly benefits if its interface specification becomes the industry standard because of the larger market acceptance of its product and its enhanced reputation.[241] Thus, a desire to sell more copies of a new program, the opportunity to reap additional profits while other software companies play "catch-up," and the minimal advantage accorded free-riders all encourage innovation of new user interface specifications independent of

238. Note that this argument would apply even if demand for the new program or interface were created by the innovation of computer hardware. Once high-resolution screens are invented, for example, it still pays the second developer to wait for the first developer to innovate a new interface specification.

239. A similar line of reasoning is used to justify giving copyright to derivative works. An author writes a novel due to the incentive of film, play, and other derivative rights: Without the possibility of these rights the author would not have bothered to even write the book. *See* Ginsburg, *supra* note 82, at 1910–11. With interfaces the situation is distinguishable: Programs need interfaces to operate, but a book is a work in itself.

240. "In fact, the effort spent designing the user interface of a computer program is usually small compared to the cost of developing the program itself." THE LEAGUE FOR PROGRAMMING FREEDOM, AGAINST USER INTERFACE COPYRIGHT 4 (Sept. 24, 1990) (unpublished paper).

241. It is interesting to note that IBM allows 15,000 third parties to write application programs for its MVS operating system. *See* Arbitration Decision, *supra* note 216, at 28. IBM Australia claims that copyright owners "have significant incentives to publish, and do publish interfaces to encourage others to write application programs for their systems. IBM, for example, has published 300–400 interfaces. . . ." IBM, submission to ACLR, Collection, *supra* note 57, at 9. IBM, along with DEC and Hewlett-Packard, established the Open Systems Foundation in 1988 in order to develop an open software environment.

copyright law.[242] Even in the absence of any copyright protection for user interface specifications, some level of innovation would be expected.[243]

3. Protection Needed to Increase the Number of Interfaces

Even if some level of innovation could be expected without legal protection of interface specifications, the advocate of such protection might argue that legal protection will result in a greater number of interfaces on the market. If a programmer is prevented from copying another's interface, the programmer will expend creative effort trying to design a better interface. This will lead to more interfaces for the community to choose from, and hopefully each will be an improvement over the last. Insisting on the use of creative effort in this case is different from the case where energy is used fruitlessly inventing around a successful, but protected, product: The users of interfaces are humans, each of whom may prefer a different way of communicating with a computer. It is not obvious until tried whether a user will like an interface, so the more created the better.[244]

In response, it may be argued that at least some degree of standardization of user interfaces is desirable.[245] Consistency in interfaces promotes ease of use and reduces the costs of retraining when new application programs are released that use a preexisting interface.[246] As more users are

---

242. The distinction between interface specifications and implementations is particularly important in this context. Protection of interface implementations is desirable even when interface specifications are not protected. A competitor could very quickly and inexpensively copy an interface implementation simply by copying the computer code itself. Allowing such copying would greatly increase a copier's price advantage in the market and would virtually eliminate the creator's lead-time advantage.

243. For example, Lotus markets its programs with the same interface in countries such as Germany, where there is no interface protection. *See also* David W. Kaye, *Colloquy on Copyright Protection of Computer Software*, 31 JURIMETRICS J. 165, 169 (1990) ("Exploiting a copyright is, no doubt, an incentive, but is is hardly the only incentive. If a competitor comes up with a significantly better interface on its program, then it can sell more of its product.").

244. Apple would not have invented and successfully marketed its Macintosh interface if it followed the industry standard or if all consumers wanted one interface. Other interfaces apart from industry standards succeed.

245. *Cf.* Landes & Posner, *supra* note 90, at 352 ("The mere fact that a particular set of symbols has become the industry standard is a tribute to the expressive skills of the particular manufacturer and should not be deemed to convert expression into idea."). *But see* Kaye, *supra* note 243, at 169 ("[I]t is not the success of the product that precludes the copyright. It is the nature of the product.").

246. A user interface is a method of communication with a computer program, not a literary work. Reading the book is the point of the book. Learning an interface is done as a means of operating a program. A familiar book is not re-read but an interface is only useful when familiar.

trained on a given computer system, more software is likely to be written for that system.[247] If interface specifications are protected by copyright law, developers other than the creator are forbidden to use any implementation of the interface's features. Such a prohibition means that only the inventor may produce products with the interface's unique features and that other developers must expend resources to create entirely new and incompatible interface specifications.[248] In contrast, if only interface implementations are protected, each developer may develop its own implementation of an industry standard user interface specification. Such a developer will only need to write and improve the implementation of a common specification. This, in turn, will reduce developers' costs, reduce barriers to entry, and provide users with a wider variety of products compatible with their existing interface.

As users invest considerable resources in training, they are less likely to change to a new product that uses a different, but superior, user interface. Many users will value compatibility over the benefits of the new standard. This may be seen as a negative effect of standardization.[249] However, to a lesser degree, permitting copyright protection of interface specifications will achieve the same result. The term of protection expires, but the other producers who were forced to develop incompatible products in the meantime will not easily change to the superior interface, or convince their customers to change, when the copyright period ends. Those who argue for a shorter period of protection for interfaces ignore the problem that users will be locked into the incompatible nonstandard interface from the moment of initial purchase.

Customers who have decided on a user interface will want a continuing supply of products compatible with that interface. If the software developer is given a legal monopoly on all implementations of an interface, it may charge monopoly prices for all new products with the interface. The customer would prefer new developers to be in competition with the original supplier. Merely allowing cross-licensing of user inter-

247. Joseph Farrell, *Standardization and Intellectual Property*, 30 JURIMETRICS J. 35, 36 (1989).

248. *But see* Lotus Dev. Corp. v. Paperback Software Int'l, 740 F. Supp. 37, 77–79 (D. Mass. 1990), where the court rejected the standardization argument, stating that the defendants could have sought a license for the Lotus interface or sold their advanced features directly to Lotus, but then said Lotus could arbitrarily refuse such schemes. It was also suggested that the defendants market their product as an "add-in," causing users to purchase two products instead of one.

249. Menell, *supra* note 197, at 1070, states that this inertia can retard innovation and slow or prevent adoption of improved interfaces."

face specifications may result in cartel-like price fixing or tacit price cooperation among competitors.[250]

Moreover, the existence of an industry standard user interface specification does not mean that there will be only one available interface. Because consumers have different preferences, there will be demand for different user interfaces. Users desiring an interface that does not use a mouse or pull down menus will create a market for that product. The result will be a proliferation of different interfaces to attract consumers with different needs. Industry standard interface specifications will also result in competition in interface implementations. Users of the industry standard interface will want faster implementations of the industry standard.

In addition, consumers who do not wish to relearn a new interface for each application program they use will benefit from the ability to purchase an implementation of a standard user interface. Such consumers will not be tied to one software developer. However, a user who decides on an interface that does not become a standard will find its network benefits diminishing as innovation in products with the standard interface increases. Finally, as described above, the developer of a user interface that becomes an industry standard will suffer little detriment and may in fact realize benefits from creating a widely used interface specification.[251]

Overall, it is a reasonable assumption that users have invested more money learning to use the interface than developers have creating it. Society would be better off allowing interface standards.[252] Many in the

---

250. *See* United States v. General Elec. Co., 272 U.S. 476 (1926) (allowing a patent licensing agreement to set a price schedule for sale of the product); United States v. Line Material Co., 333 U.S. 287 (1948) (holding that two patentees cross-licensing their interdependent patents to secure additional mutual benefits violates the Sherman Act); United States v. United States Gypsum Co., 333 U.S. 364, 400–01 (1948) (holding that industry-wide license agreements under which price control was exercised established a prima facie case of conspiracy). *Cf.* Kaplow, *supra* note 84.

251. One solution to encourage standardization and still provide an incentive to produce is the compulsory license. It is a solution somewhat similar to that arrived at in the IBM-Fujitsu arbitration. *See* SCHERER, THE ECONOMIC EFFECTS OF COMPULSORY LICENSING (1977); Venit, *Technology Licensing in the EC*, 59 ANTITRUST L.J. 485, 496 (1991). In antitrust litigation, compulsory licensing of patents is an available remedy. *See* Hartford-Empire Co. v. United States, 323 U.S. 386, 417 (1945); United States v. Glaxo Group, 410 U.S. 52 (1973); *see generally* AREEDA & KAPLOW, *supra* note 80, ¶¶ 190, 284.

252. For an economic analysis of the tradeoff between production costs and consumer value, see William W. Fisher, III, *Reconstructing the Fair Use Doctrine*, 101 HARV. L. REV. 1659, 1703–04 (1980).

computer industry do not want interface protection.[253] They believe that use of an industry standard would be more beneficial for everyone.

D. Applying the Proposed Test

Analysis of the justifications for copyright protection of interface specifications indicates that copyright law should not provide monopoly power to the creator of a new specification. Although protection for interface specifications might increase the incentives for innovation, such innovation would continue even without legal intervention. Moreover, refusing to protect interface specifications would allow development of industry standard interfaces, with all the attendant benefits of standardization.[254]

These conclusions do not dictate that no protection be afforded to user interfaces. Instead, the proposed test focuses on expression. The interface specification is a set of rules, independent of expressive content until implemented in a particular way. Thus, the proposed test would not extend copyright protection to interface specifications. The interface implementation, however, is composed of computer code. It is a specific expression of the ideas embodied in the specification and should therefore receive copyright protection.[255]

Therefore, if a software developer obtains a computer program from a rival and decides to integrate the interface into its own program, it could carry out the following procedure[256] without infringing the other

253. For example, firms such as Unisys (advocating specific exemptions in copyright legislation for interface specifications), Sun Microsystems (submitting that the "look and feel" of a program should not be protected by copyright), Fujitsu (arguing for an explicit "interface" exclusion), Bull HN Information Systems (recommending that interface specifications should not be protected by copyright and copying of these specifications should be permissible), McDonnell Douglas (concluding that extension of copyright protection to specification of interfaces would have a devastating effect on industry development). Submissions to ACLR, Collection, *supra* note 57. *See also* Pamela Samuelson & Robert J. Glushko, *Comparing the Views of Lawyers and User Interface Designers on the Software Copyright "Look and Feel" Lawsuits*, 30 JURIMETRICS J. 121, 121 (1989) (79% of industry respondents opposed to "look and feel" protection).

254. For a summary of the adverse impacts of permitting protection of user interface specifications, see Menell, *supra* note 197, at 1071.

255. A similar analysis would be used in applying the proposed test to any individual element of the user interface. A screen display, for example, is part of a user interface. However, it is an element of the specification because the specification dictates the layout of elements onto the screen. Thus, the screen display itself is not protected. Under the proposed test, if programmer A uses a screen design from programmer B, programmer A has not violated programmer B's copyright.

256. A clean room procedure is used to develop a clone of a program where the programming team independently develops a complete program. The procedure described above is different from the expensive clean room described in David S. Elkins, *A Guide to Using "Clean Room" Procedures as Evidence*, 10 COMPUTER L.J. 453, 480 (1990). *See*

developer's copyright. The developer wishing to create a new interface implementation would create two teams of programmers. One team would decompile the rival's program and determine the interface specifications. This team would then pass the specifications on to the second team. The second team would, without any knowledge of the interface implementation used by the rival, code an implementation of the rival's interface specification. The new implementation could then be integrated into new programs.[257]

This procedure would pass the proposed test since only the non-protected interface specification is duplicated; the protected interface implementation is not. This solution does result in some inefficiencies. If standardization is beneficial, why have firms independently exerted effort to recreate an existing interface implementation? The answer is pragmatic. There must be incentive to motivate creation of new user interface implementations. Developers spend time and effort expressing an interface in error-free code. If that code is protected, the first firm to innovate a new interface specification will get a head-start in the market.[258] Rivals will incur costs in coding and testing new implementations of the standard interface specified and will receive decreased price advantages over the specification's creator. Application of this solution fits neatly into the copyright scheme, as all computer code can be equally protected. It is a solution that balances the need to provide incentives for development with the desire for the benefits of standardization.

E. Conclusion on Copyright

This Section has proposed a scheme to solve copyright problems involving computer programs. It is a scheme that can be applied to the current problems in this area: protection of interfaces, output, and function. It can be applied in Australia, and also in the United States instead of the merger doctrine.

To consolidate, the scheme is as follows. Computer programs should

---

also Jorge Contreras et al., Recent Development, NEC v. Intel: *Breaking New Ground in the Law of Copyright*, 3 HARV. J. L. & TECH. 209, 218–21 (1990) (describing the costs of clean rooms).

257. Similarly, if a programmer designed a program with expression substantially similar to another interface without any knowledge of it, there is no copyright infringement. *See also* Conference, *Last Frontier Conference Report on Copyright Protection of Computer Software*, 30 JURIMETRICS J. 15, 23 (1989).

258. *Cf.* Vance F. Brown, *The Incompatibility of Copyright and Computer Software: An Economic Evaluation and a Proposal for a Marketplace Solution*, 66 N.C. L. REV. 977, 1009 (1988) (Protection of software should provide monopoly protection only for the developer's legitimate lead time.).

be separate works under the Copyright Act.[259] Computer programs can only infringe the copyright in other computer programs. Only computer programs can be derivative works of computer programs. If a form of expression is not detailed enough for a computer to execute it without further human intervention, it is not a computer program. To determine if there is copying, the expression to be examined is the algorithm in the form of computer code. Function, output, and specifications are to be ignored. Literal copying is infringement.[260] A low-level version of the code that a computer uses to execute the program is protected: Copying this would infringe the copyright in the programmer's code. When examining code expressed in different languages, the algorithms expressed as code at a level of abstraction comparable to that chosen by the programmer to express the algorithm must be substantially similar. If different algorithms are used to achieve the same result, or if the algorithms are similar only when expressed at a level far higher than the code at issue, then there is no infringement. These tests assist in the application of the idea-expression distinction to computer programs. In the end, in hard cases, the question is one of degree: At what level of abstraction should the algorithms be examined to see if they are the same? The dis-

259. *But cf.* Glynn S. Lunney, Jr., *Copyright Protection for ASIC Gate Configurations: PLDs, Custom and Semicustom Chips*, 42 STAN. L. REV. 163 (1989) (There should be no difference in hardware and software protection so the market can control levels of investment in each.).

260. Menell, *supra* note 197, at 1082, concludes that "legal protection for application programs should not extend much, if at all, beyond protection against literal copying, except for new, useful, and nonobvious improvements." The main reason for such protection, he states, is similar to that given above: to ensure that the lead time will be significant to recover development costs. *Id.* at 1086. Instead of extending protection beyond literal copying to copying of the underlying algorithm, Menell has imported the patent standard into copyright law. The problems that patent law faces using this test have been discussed above. Importing the patent requirement of novelty and nonobviousness is described as "simpleminded" in Wiley, *supra* note 34, at 145. *See also* Fred Fisher, Inc. v. Dillingham, 298 F. 145, 150 (S.D.N.Y. 1924) (Hand, J., distinguishing copyright from patent).

Menell elaborates the test, stating that limiting copyright to expression means that the expressive aspects of the structure of the program that are not functional attributes should be protected. The court is required to separate the functional aspects from a program's expressive aspects. Menell, *supra* note 197, at 1085. Under this test, only comments directed to the programmer would be protected—all error-free computer programs are functional. Menell says his test is consistent with copyright tests for architectural plans, business forms, and game rules. *Id.* at 1085 n.231. This is not so. Architectural works are protected by copyright, as are forms if they convey information. These works are functional. Rules of a game, as long as they are written down, are copyrightable regardless of whether they describe a game that is functional, efficient, fair, or foolish. Expression that allows a function to be carried out, such as a recipe or a computer program, is copyrightable. Copying the result of carrying out the instructions, the cake or the user interface, is not an infringement of the instructions.

tinctions that a judge would have to make would be no different from those made in deciding other difficult copyright cases.

CONCLUSION

This Article has examined patent and copyright protection of algorithms expressed as computer programs. The focus was on applying a knowledge of algorithms to issues currently in dispute. It was assumed that protection of some sort is needed for computer programs' intellectual components. As copyright is the preferred vehicle for protection internationally, and has been so for a number of years, the scope of copyright protection was examined. The idea-expression distinction was applied to determine what should be protected as a computer program and what should constitute copying.

The inquiry into patent law was different. The Article examined the objections currently made by software developers to the patenting of computer programs and concluded that those objections were no different from objections that could be made to the patent system generally, albeit in a more extreme form in some instances. Secondly, as a computer program is both expression of a process and the means to carry out the process, the Article showed that the patent rules come close to protecting what copyright protects, expression. If high-level algorithms are not protected (being abstract ideas) and protection is not given to algorithms expressed directly in computer code (being the domain of copyright), then patent is left to protect algorithms where a computer is the desired processor but the algorithm is expressed in such a way that it could be used in many programming applications and various programming languages. The court would then have to determine whether a program's coding used that algorithm. On this Article's proposed test of substantial similarity, the courts would use a similar analysis to decide whether there is breach of copyright. In other words, copyright and patent would cover the same subject matter. Copyright does it more efficiently.

No patent protection should be provided for computer software. Copyright protection is adequate. It would be inefficient to have dual coverage of one product to achieve the one goal of promoting innovation. Protection is needed to stop rivals from taking the intellectual effort in the software created and using it in a similar product sold at a reduced price. The copyright system can more efficiently and fairly provide the protection needed.

First, the copyright system has fewer formalities. No registration or disclosure is required. There is no waiting period.

Secondly, the copyright system is fairer. There is no fight to determine who was first. If two people independently code a program in the same way, as is likely to occur in coding algorithms in computer languages, both are protected. No monopoly is granted, which in a rapidly expanding field seems like a lottery prize to the lucky programmer with the best lawyer. People are encouraged to create, knowing that what they create will be protected.

Thirdly, the patent system cannot efficiently decide whether an algorithm or program is novel. Most programs written are not novel and are obvious. Less than one percent of computer programs, it is claimed, are patentable.[261] It would be an expensive and far-reaching inquiry to determine if an algorithm is novel and nonobvious, as algorithms have existed for centuries and are used in a variety of fields. With so much to examine, a wrong result is likely in many cases. Assume that a limitation was put on the search, so that only the application of the algorithm had to be novel. To computer scientists, using an algorithm in a computer program is always obvious. Additionally, protection would be given to the application of the algorithm, in this field, the expression of the algorithm as a computer program. Isn't that what copyright protects?

Fourthly, programs have a short life, but the algorithms used in a program can be used repeatedly in a variety of applications. Algorithms are the building blocks of computer science. Lock up algorithms and development will cease. One need only protect the product and not the tools in order to encourage creation of the product.

Fifthly, if new algorithms continue to be discovered as long as science endures, what need is there for incentive? The incentive should promote application of the algorithm. If applied in a computer program, the expression is protected. Inventions and processes do not "contain" any expression for copyright to protect, so alternative protection, generally that of patent, is required. For example, in the creation of a better mousetrap there is no expression to protect. The opposite is always the case for computer programs.

The copyright regime gives adequate protection to encourage innovation and reward inventors. What is valuable in software is its use; unlike in other areas, the form of expression is used directly to perform a task. There is no need to give additional protection to the algorithm if the most valuable form of the algorithm, the computer program, is protected.

---

261. *See* Duncan M. Davidson, *Protecting Computer Software: A Comprehensive Analysis*, 23 JURIMETRICS J. 339, 357 (1983).

Copyright does not go too far, so as to protect function. To do so would be to provide patent-like protection with less stringent tests. As concluded in the user interface area, some protection is needed, and that is provided by giving protection to the expression implementing the interface. As can be seen in the large amount of public-domain software available, over which no copyright is asserted, providing copyright protection to software to encourage innovation may be erring on the side of caution.

The legal rules necessary to implement this proposal are simple. Copyright will be given to computer programs as a separate category of works within the copyright system. Patent coverage will be denied for such programs. Any algorithm expressed as a computer program will not be patentable. An algorithm may be patented where no software is involved. That patent cannot be infringed by using the algorithm in a computer program. The simple result: Patent law will protect hardware, and copyright law will protect software.