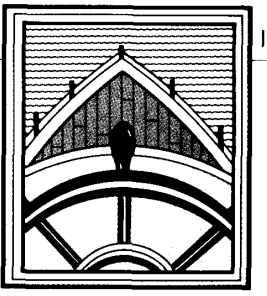


Is Copyright Law Steering the Right Course?

Pamela Samuelson, *University of Pittsburgh*

A landmark copyright decision questions the legality of reusing the abstractions and structures underlying software. Although well-intentioned, this flawed decision could harm software-engineering practice.

Good software-engineering practices may now be illegal under copyright law. Suppose you have just left your job as a software engineer for a major company to set up your own business. For your old firm, you once developed an inventory-control program in Cobol for a steel manufacturer. Now a client comes to you and asks you to develop some Ada software for control of his auto-parts inventory. You don't have the source code for the steel-manufacturer program. Even if you did, you would be reluctant to use it because, for one thing, it wouldn't be good software-engineering practice, and second, you would probably sense that that code belonged to your previous employer. But surely you could reuse some of the structure and data abstractions that worked so well in the steel-manufacturer program.

It may be good software-engineering practice to reuse the structure and data abstractions, but it may also be illegal — at least if the steel-manufacturer program is copyrighted. Whether it is illegal depends on which part of the country you live or work in, for judges in some jurisdictions have decided that structure and other high-level design features may be protected under copyright law. Judges in other jurisdictions have reached the opposite conclusion. The box on p. 80 examines some contradicting opinions.

I focus on one Third Circuit Court of Appeals decision that concluded that structural aspects of programs *are* protected by copyright law. (The Third Circuit includes New Jersey, Pennsylvania, and Delaware.) This 1986 case is known as the Whelan decision after its plaintiff, Elaine Whelan.

Why focus on the Whelan case? There are three reasons.

First, the three appellate judges in the Whelan case proposed and then applied a very far-reaching test to determine what constitutes copyright infringement in

0740-7459/88/0900/0078/$01.00 © 1988 IEEE

software cases, a test so broad that it would make even similarities at an algorithmic level a basis for copyright infringement.

Second, the judges developed an elaborate rationale to use this test that relied heavily on a theoretical model of software as a literary work (like a novel) when it should have analyzed the copyright issues based on a model of software as a functional work (like an engineering drawing). (The box on p. 83 explains these types of works.) Nor did the judges consider patent protection, which I believe is a better model for software protection. To preserve the appropriate balance between copyright and patent protection for functional works, much more borrowing (what software engineers call "reuse") is tolerated in copyright cases involving functional works than in literary works. I believe that balance was ignored in the Whelan case.

Third, although the judges in one other appellate case decided not to follow the Whelan case or apply the Whelan test, judges in several other cases have cited Whelan approvingly or have followed its lead, which means that the decision most dangerous to good software-engineering practices may become the law of the land. Two of these cases — Digital Communications Associates, Inc. v. Softklone Distributing Corp. (US District Court, Northern District, Georgia, 1987) and Broderbund Software, Inc. v. Unison World, Inc. (US District Court, Northern District, California, 1986) — involved whether screen displays are covered by copyright, a controversial proposition that both courts accepted.

When judges address a new kind of copyright problem, such as whether the logic and structure of software should be protected by copyright, they first try to find the closest similar problem from earlier cases. If this investigation is inconclusive, the judges then try to decide whether protection in the case at hand (and in others like it that might arise later) will promote the ultimate goals of the copyright law. The major aim of the copyright law is to promote progress in science and the useful arts.

The judges in the Whelan decision were persuaded that copyright protection for software structure would promote innovation in the software industry. But they may have been wrong. Judges in future software-copyright cases might be persuaded that copyright protection for software structures would more likely retard than promote innovation in the software industry. If so, they would have grounds for rejecting the Whelan approach.

The judges were persuaded that copyright protection would promote innovation in the software industry. But they may have been wrong.

Obviously, the answer to the question of what effect copyright protection for structural elements will have on innovation depends in part on what "structural elements" is interpreted to include. The Whelan decision is dangerous not just because it would make virtually any structural similarity between programs grounds for copyright liability but also because no other court decision has yet articulated a narrower rule with which you can sort out what structures can and cannot be protected. However, because it is difficult to identify precise levels of structural abstractions that can be protected by copyright instead of by a patent, it may be very difficult or even impossible to develop a narrower test than Whelan's.

Because the Whelan decision has such a potentially drastic effect on the future of software-engineering practices, programmers must be aware of the decision and its implications and should help decide whether such strong protection for structural features will promote or retard innovation.

I argue that protecting the logic and structure of programs through copyright law, except perhaps at very low levels of abstraction, presents a serious threat to innovation and would be a bad thing for both copyright law and software engineering.

If high-level abstractions, such as logic and structure, do receive intellectual-property protection, the better method is to use patent law, not copyright law, because patent law has a higher creativity standard, shorter duration, and proven track record of promoting technological progress.

Copyright versus patent

In 1980, Congress extended copyright law to include software, marking the first time a technology had been covered by copyright. Traditionally, technologies have been protectible only under patent law. When Congress put software into the copyright system, it likened source code to the text of a book and object code to a mere copy of the source code — without realizing that software was a technology and thus protectible by patent law.

Copyright. Copyright protection is available for original works of authorship that have been fixed in some tangible medium of expression, according to Title 17 of the US Code, section 101 and subsequent sections. The protection begins when the work is first put into a tangible form and lasts for the life of the author plus 50 years (or 75 years from publication for a corporate author). The excep-

Copyright decisions on software structures

Four court cases have explicitly addressed whether structural aspects of software are protectible by a copyright on the software. Two (including the Whelan case described in the main text) have found software structures to be protectible; two have found software structure not to be protectible.

Synercom case. Synercom Technology, Inc. v. University Computing Co., which was decided by a Texas federal trial judge in 1978, was the first software-copyright case to consider structural similarities as a basis for infringement. The precise issue was whether similarities in the sequence and ordering of input formats was a reasonable basis for finding copyright infringement.

Synercom, the plaintiff, had developed a structural-analysis program for engineers. The Synercom user manual described the program's input formats. Engineering Dynamics used those input formats in its own preprocessor program, which University Computing commercially distributed. Synercom's lawyers showed that while there were hundreds of structural-analysis programs available, only Synercom's and Engineering Dynamics' used the same formats. Moreover, it was clear that Engineering Dynamics had adopted these formats to better compete with Synercom's program.

Engineering Dynamics raised two defenses to Synercom's input-format infringement claim. First, relying on a set of cases that had ruled that blank forms were not proper subject matters for copyright (because blank forms do not themselves communicate information but only receive information supplied by users), Engineering Dynamics contended that Synercom's input formats were not copyrightable. The court rejected this claim, saying that "these input formats express to the user the sequencing of data for simplified access to the software. The formats by their placement of lines, shaded art, and words tell the user what data to place where and how to do it. It communicates the selection, arrangements, and the sequences."

But Synercom lost the case over Engineering Dynamics' second defense: that it had taken no more of the Synercom formats' expression than necessary to take the idea. Engineering Dynamics relied on a series of copyright cases that had ruled that, when the ideas in a work were inextricably connected with the expression, it was acceptable for others to take the expression so they could take the idea.

Engineering Dynamics convinced the judge that Synercom's input formats were like the "H" pattern for car stick shifts. This pattern may have been chosen randomly from several alternatives and could be expressed in many ways (such as a prose description, a diagram, or a photograph), each of which might be copyrightable, but that would not mean that the first car manufacturer to copyright a diagram of the "H" pattern could prevent other manufacturers from using this pattern in their own cars or diagrams of a stick-shift pattern, since the "H" pattern was an unprotectible idea in the copyrighted diagram. The judge could not discern what idea there might be in Synercom's input formats if not their order and sequence, so he ruled that Engineering Dynamics had taken only the idea in Synercom's work and so had not infringed the copyright.

SAS case. The next case that considered structural similarities as a basis for software-copyright liability was the SAS Institute, Inc. v. S&H Computer Systems, Inc. case decided by a Tennessee federal trial court in 1985. SAS markets a well-known statistical-analysis program that runs on IBM mainframes. S&H developed a version of the SAS program to run on Digital Equipment Corp. VAX computers after obtaining a copy of the SAS program and its source code by a license from SAS.

Although the judge in the SAS case relied on evidence of extensive structural similarities (of an unspecified sort) between the two programs in finding that S&H had infringed the SAS copyright, this case is not a strong precedent on the protectibility of program structures because it also involved the exact copying of portions of the source code, attempts by S&H to disguise the extent of its direct copying from the SAS code, the making of multiple copies of the code to reverse-engineer it in violation of the license agreement, and other questionable or inequitable conduct on S&H's part.

Plains Cotton case. Like the Whelan case described in the main text, the Plains Cotton Cooperative Association v. Goodpasture Computer Service, Inc. case involved charges of copyright infringement based solely on structural similarities between software programs. However, the Fifth Circuit Court of Appeals (which includes Texas, Mississippi, and Louisiana) reached a different conclusion in this 1987 case than the Third Circuit had in the 1986 Whelan case. (Appeals court decisions are binding on all federal courts in their circuit. When appeals courts in different circuits disagree, the Supreme Court must eventually resolve the difference, which it has not done yet in this case.)

Plains Cotton claimed that Goodpasture used design specifications that Plains Cotton owned in developing a competitive cotton-marketing program. Goodpasture had obtained these design specifications, Plains Cotton alleged, by hiring four former Plains Cotton employees who had developed Plains Cotton's mainframe program and who had done design specifications for a PC version of it.

Although the appellate court agreed that there were significant organizational similarities between the two programs, it concluded that market factors played an important role in determining the sequence and organization of cotton-marketing software. Such organizational similarities were, therefore, unprotectible ideas of the software, not expression. The court in Plains Cotton expressly declined to follow the Third Circuit's Whelan decision or apply its test for software-copyright infringement. It did not cite the SAS case, but it applauded and relied on the reasoning in the Synercom case.

Confused status. In 1987, the Supreme Court declined to hear the appeals of the losing parties in the contradictory Whelan and Plains Cotton cases, leaving uncertain whether (or to what extent) structural similarities between programs are bases for copyright infringements.

Although these decisions are only binding in their own jurisdictions, they have spawned additional litigation in others. It is no coincidence that Lotus Development Corp.'s lawsuits against Mosaic Software and Paperback Software, claiming copyright infringement because the defendants market competitive spreadsheet programs with similar look and feel, was filed within hours of the Supreme Court's decision not to hear the Whelan appeal. In March, Apple Computer sued Hewlett-Packard and Microsoft Corp. for copyright infringement of its screen designs, also relying on the Whelan decision.

Until the Supreme Court does decide to hear a program copyright case, software engineers in other than the Third and Fifth circuits can't be sure what the law is on this important issue.

tion is if the work is published without a copyright notice, in which case the work is in the public domain (it becomes public property).

The copyright gives the work's owner the right to exclude unauthorized people from making and distributing copies of it. Authors are given these rights as an incentive to be creative and to make the fruit of their creative labors available to others.

The fundamental principle of copyright is that it protects only the expression in a work, not the ideas in it. You can protect only the expression through copyright. Some things that copyright law considers as unprotectible ideas can be patented. Other things might be eligible for protection as trade secrets, but copy-

right law will not protect them.

Of course, "expression" and "idea" are broad terms that the courts have refined over the years. For example, "expression" includes more than the exact words of the copyrighted text. As one judge in a frequently cited case (Nichols v. Universal Pictures Corp., US Appeals Court, Second Circuit, 1930) put it, "It is of course essential to any protection of literary property ... that the right cannot be limited to the text, else a plagiarist would escape by immaterial variations." Thus, two works need not be identical, or even nearly identical, for a court to find copyright infringement. Infringement can be found if there is "substantial similarity" in expression. How much more than the exact words will be considered expression depends largely on the nature of the copyrighted work (see the box on p. 80).

The meaning of "idea" likewise includes more than the general abstract concepts in a work. The copyright statute itself sets forth an illustrative list of those things that copyright law considers to be unprotectible ideas. "In no case," the statute says, "does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is embodied in such work." The list is only illustrative, and courts have also found facts, mathematical formulas, and functional aspects of engineering drawings to be unprotectible ideas.

A century-old case, Baker v. Selden, is the most famous explication of the idea/expression distinction and its implications for functional works. Selden had developed a new kind of accounting system and had written books about it that he copyrighted and that included some sample ledger sheets. Baker wrote a book about Selden's accounting system that included sample ledger sheets that were very similar, although not identical, to those in Selden's books. Selden's heirs sued Baker for copyright infringement.

But the Supreme Court ruled in 1879 that Baker's work did not infringe Selden's copyright because it recognized a distinction between the "practical art" that a book might describe and the way in which the description might express it. To protect the practical art, the court said, you would have to have a patent. The copyright could only protect how the practical art was described. Furthermore, the court said, any similarities in expression between the two works that were attributable to their being about the same practical art (here, the same bookkeeping system) could not serve as the basis for copyright liability — otherwise, the copyright would end up protecting the system. Because Baker's book was similar to Selden's only in this way, it did not infringe the Selden copyright.

From this case emerged the copyright rule that although such things as engineering or architectural drawings might be copyrightable as drawings, the copyright in the drawings did not protect the engineering or architectural design of the functional thing depicted in them.

No matter how creative the design might be, copyright law considers the engineering design to be the work's unprotectible idea.

No matter how creative the design might be, no matter how elaborately it might be articulated, copyright law considers the engineering or architectural design to be the work's unprotectible idea.

That does not mean that a highly creative engineering design cannot be protected by intellectual-property law, just that it can only be protected by another law: patent law. The only thing in an engineering drawing that the copyright will protect is the drawing as a drawing.

Patent. Patent law includes technologies in its subject matter by providing that any new, useful, and nonobvious process, machine, manufacture, or composition of matter, or any improvements of these kind can be patented.

The creativity level for an innovation to be eligible for a patent is high. To be eligible, an invention must be such a significant innovation that it would not be obvious to someone skilled in the art who tried to make such a thing. By contrast, the copyright standard of creativity is very low. Something must only "owe its origin to its author" and not be copied from another source to be copyrightable.

Patent protection does not arrive automatically at the time of the invention, as copyright protection does for writings. Instead, you must apply for a patent from the federal Patent Office and undergo a rigorous examination by patent examiners to determine if patent's high creativity standards are met. Once a patent is issued, it lasts for 17 years.

Despite its shorter duration (17 years compared to copyright's 75 years for corporate authors), patent protection has traditionally been considered stronger than copyright for two reasons. First, a patent protects the innovator against anyone else's use of the same or equivalent thing, even if the second user developed it without knowledge of the patented invention. (Under copyright independent development is a complete defense against infringement claims.) Second, patent law covers many of the things that copyright law considers to be ideas, such as processes and engineering designs.

The Whelan decision

In the Whelan decision, the Third Circuit Court of Appeals went a step further than Congress did by treating program logic and structure as protectible by copyright.

In 1978, Rand Jaslow hired Elaine Whelan to develop a program that he could use to manage his dental-laboratory business. He paid the full cost of the development and gave Whelan, who had no experience with dental-laboratory businesses, extensive access to his business. Both expected the program to be marketed to other dental labs, and Jaslow agreed to let Whelan retain the copyright but insisted that he get a 10-percent royalty on all copies distributed to others.

A year or so later, Whelan delivered the Dentalab program to Jaslow. It was written in the now-obscure Event-Driven Language and ran on an IBM mainframe. For about two years, Jaslow acted as a marketing agent for the Dentalab program and

so received a 35-percent royalty in this period.

During 1982, Jaslow decided that there would be a good market for a program like Dentalab that would run on an IBM PC and that was written in a much more commonly used language like Basic. Entrepreneur and self-taught programmer that he was, Jaslow undertook to write such a PC program in Basic. In 1983, he began marketing his Dentcom program at a price considerably less than the $10,000 Whelan was charging. Whelan responded by suing him for copyright infringement.

Arguments. There was no question that Jaslow had access to and studied some of Whelan's source code (which he paid for) in preparing his competitive program, but he did not copy portions of Whelan's code. Ironically, Whelan derived a lot of the data and file structures for how her program performed functional tasks from examining Jaslow's files and observing how his staff did things. Jaslow also helped Whelan design the user interface. Losing the lawsuit meant that Jaslow might be foreclosed from automating his own method of doing business because Whelan automated it first.

Because the two programs were written in different languages, there were, of course, no exact similarities in the source code of the two programs. Neither the trial-court judge nor the appellate court considered Jaslow's Basic program to be a line-by-line translation of Whelan's Event-Driven Language program. Indeed, the trial judge remarked that a line-by-line translation of a program from one language to another would be very inefficient and might well be difficult or impossible to do.

A more efficient way to copy an Event-Driven Language program into Basic, the judge noted, would be to "study the method and manner that the computer receives, assembles, holds, retrieves, and communicates data." This, in turn, "requires a study of the manner in which information flows sequentially from one function to another." Once this has been done, the judge said, you could then "copy this exact manner of operation for use in a computer that responds to commands written in a different source language." By this description, the judge implied that it was by this process of abstraction that Jaslow copied Whelan's work.

To explain why he found Jaslow's program to have infringed Whelan's copyright, the judge said that Whelan's copyright covered "the manner in which the program operates, controls, and regulates the computer in receiving, assembling, calculating, retaining, correlating, and producing useful information on a screen, printout, or by audio communication," again implying — but not directly saying — that the similarities in the Whelan and Jaslow programs were of this functional sort.

Whelan's lawyers argued that if a novel's structural aspects were covered by copyright, so should software. The appellate court agreed.

Although both the trial judge and the appellate court were impressed by similarities in the programs' screen displays, both considered these similarities to be evidence of similarities in the copyrighted underlying program text: Both sets of judges found the copyright on the underlying program text had been infringed, not the screen displays themselves. The judges did not seem to appreciate that differently structured programs could have very similar interfaces or that similarly structured programs could have different interfaces.

The trial judge had been unclear about what kinds of similarities he had found between the programs. The appellate court decided that the real issue in the case was whether similarities in the structure, sequence, and organization, such as in data or file structures or in subroutine actions, could be the basis for infringement. (The trial judge had made no reference to there being any structural similarities between the two programs.)

Jaslow's main contention on appeal was that the features of the Whelan program that he copied were part of the "process, procedure, system, or method of operation" of the copyrighted software — things that the copyright statute says are unprotectible ideas.

Whelan's main contention on appeal was that, in earlier cases involving novels and dramatic plays, structural features of copyrighted works had been held to be protectible expressions. Whelan's lawyers pointed out that copyright law classifies software as a literary work, just as it does novels. If it made sense to say that structural aspects of novels and plays were protectible expression, Whelan's lawyers argued, it should make equal sense to apply this rule to software. The appellate court agreed with Whelan's lawyers.

New test. Although confident in its judgment that infringement had occurred, the appellate court was troubled by how to distinguish ideas from expressions in software cases. Both to explain its decision and to provide guidance to other judges, the appellate court developed and applied a new test to distinguish ideas from expressions in software cases.

The unprotectible idea in software is its "general purpose or function," the court said. Everything else about the program is protectible expression, it said. The only exception to this "everything else is expression" rule would be when there were only one or a very few ways to structure the software to perform the general function. If there were only one or a very few ways to structure the program, the structure would be considered to be part of the software's idea.

The court likened its analysis to that in Baker v. Selden, where Baker escaped copyright-infringement liability because the court ruled that similar ledger sheets were necessary if both Baker's and Selden's books were to show how to practice Selden's method of bookkeeping.

Applying this test, the appellate court found that the idea in Whelan's program was its general purpose or function: the computerized management of dental-laboratory businesses. Jaslow was free to use this idea without fear of liability to Whelan. But he had taken more from

Whelan's program than this idea, the court said. He had also used some of the data and file structures and some of the subroutine functions that Whelan had used, it said, and these structures and functions were part of the expression of Whelan's program because the use of those structures and functions was not necessary to manage a dental-laboratory business. Thus, Jaslow's use of them did not fall under the test's "everything else is expression" exception, and the court found Jaslow had indeed infringed Whelan's copyright.

Decision critique

It is easy to criticize individual aspects of the Whelan decision and its controversial test for software-copyright infringement. If taken literally, the test would direct the courts to expect, in general, to be able to find only one idea per program — its general purpose or function — regardless of the program's complexity. All else about the program would generally be protectible expression — presumably including the algorithms used to implement the program's functions, although even a cursory study of the copyright statute and case law would leave little doubt that algorithmic similarities are the same as idea similarities and thus not protected. (Patent law, at least in cases so far, seems to exclude algorithms.)

It is more difficult, however, to say just where the judges in the Whelan case went wrong in their analysis, why they did so, and what they should have done instead. What flaws in their conceptualization of the problem led them to adopt such an absurd test?

Patent versus copyright. The major problem with the Whelan decision and its test is that it fails entirely to consider the proper relationship between patent and copyright protection for software.

Software is the first technology that has ever been copyrightable. Until software came along, technologies and other utilitarian works were protectible only by patent law. A work had to have utility to be patentable, and it could not have a utility (beyond conveying information or displaying an appearance) to be copyrightable. Without clearly understanding the problems that might result from putting a technology into the copyright system, Congress added software to copyright law's purview in 1980 — making it both patentable and copyrightable. Unfortunately, Congress did not say what it intended the relationship between copyright and patent protection for software to be. The Whelan decision ignored the issue altogether.

Had the judges in the Whelan case considered a little harder both the copyright statute's mandate that no "procedure, process, system, [or] method of operation" be protectible by copyright and the decision in Baker v. Selden, they might have realized that the reason copyrights do not protect processes is because processes and the like are patentable.

Under Baker v. Selden, what patents may protect is what copyright law considers to be ideas. At the very least, a test for software-copyright infringement should provide a way to distinguish between the software's process, which copyright cannot protect, and the other, nonliteral features of the work that might be protected as expression under copyright.

Abstraction levels. In searching for a way to distinguish between the potentially patentable process and the potentially copyrightable nonliteral similarities, judges considering software-copyright infringement cases should recognize distinctions among higher and lower level structural abstractions. It is much more consistent with copyright tradition to protect very low-level abstractions, such as the instruction-by-instruction sequence of the source code, than to protect higher level abstractions.

Because there is no generally agreed-upon scheme to depict how many levels of abstraction might exist in any piece of software, because different people might put different things in different levels, and because those who develop software

Copyright types

There are three basic categories of copyrighted works: artistic works, factual works, and functional works. Artistic works generally enjoy the broadest copyright protection, factual works a narrower scope, and functional works a very narrow scope. (Truly functional works, such as chairs or microwave ovens, are because of their functionality not protectible under copyright at all.)

In general, more elements of artistic works (like a play) will be considered to be expression than in a historical work (like a biography), and more things will be considered to be ideas in functional works (like an engineering drawing) than in an artistic or factual work.

The underlying rationale for this variance is that different groups need different amounts of protection.

The law gives artistic works the most protection to encourage artists to seek ever more diffuse ways of expressing themselves on themes of enduring interest (the 10,000th novel about love may still teach us something about love, particularly if the law forces it to present it differently than the previous 9,999 novels).

Factual works fall in the middle because historians, for example, need to be able to draw more on each other's work to advance the progress of knowledge.

Functional works have the least protection because bridge engineers, for example, have a great need to draw on the work of other engineers who have designed good bridges so they in turn can build safer and more reliable bridges. Furthermore, patent protection is available to reward truly innovative engineering designs.

Software initially seemed to be like a literary work to Congress because it is written out in source code, just like a draft of a book. But concentrating on this aspect of software obscures its functional aspects and the engineering process by which it is developed. It is more appropriate to think of software as akin to an engineering drawing than to a novel.

tend to think about levels of abstraction as essentially equivalent to each other, it may be impossible to formulate a test that will be precise enough to say what abstractions copyright law will protect, what abstractions patent law will protect, and what abstractions both laws should consider to be unprotectible ideas.

Such a test is what software engineers need for adequate guidance about what abstractions can lawfully be reused. But the Whelan decision did not even try to make any distinctions based on levels of abstraction.

Although the judges in Whelan were presented with arguments that software was different from other literary works and that progress in the software field might be impeded if structural abstractions and how subroutines functioned were protected by copyright, the appellate court saw no reason to assume that the conditions necessary for progress in the software field were any different from those for other literary works. "If protecting structural features of traditional literary works had promoted literary progress, why should it not do so for software as well?" they reasoned.

How to handle software

The questions raised by the Whelan decision deserve serious attention. Because the purpose of the copyright law is to promote the progress of science and the arts, the fundamental issue is whether protecting the logic and structure of software through copyright law will advance progress and innovation in the software field.

Arguments for copyright. Those who say that strong copyright protection for software structures will promote innovation in the software industry might argue that a fundamental premise underlying the copyright system is that granting temporary property rights in innovations induces people to be creative and to invest their money and energy in innovation. If others could freely appropriate the fruits of the innovators' intellectual labor, innovators would be deterred from investing in innovation. A copyright limits the power of others to appropriate that intellectual labor and creates the financial base to recoup the investment by granting the innovator exclusive rights.

By protecting the object code of the program from unauthorized duplications and commercial redistribution, copyrights clearly deter outright piracy of the code, the argument for protection continues. But if copyright protection for software was limited only to the source code, too little of what was valuable about the software would be protected and investment in software would be deterred, retarding innovation.

Coding, as the judges in the Whelan decision observed, is a trivial exercise. The real intellectual labor in software development comes at the design phase, which includes designing the structures for files and data, the arrangement of subroutines, and the organization of functions. It is this intellectual labor that represents the value that copyright law should protect, the argument for protection concludes.

Because patents are now issued for high-level software designs, there should be no need for copyright to fill this role.

The major flaw with this argument is that it ignores that there is another body of intellectual-property law — patent law — besides copyright to fill the gap where protection is needed. Although patent law may have gotten off to a bad start as a way of protecting inventive software and software designs, more recent case law and US Patent Office practices suggest that patents are becoming an increasingly significant source of intellectual-property protection for software innovators.

How patents work. Since a 1981 Supreme Court decision in Diamond v. Diehr, software patents have become increasingly common. Because patents are now issued for high-level software designs, such as the organization of software's functional components, there should be no need for copyright to fill this role in software protection.

Reflect on why software engineering as a discipline has arisen and why it has sought to promote reuse as a goal in software design. Over the years, software designers have come to realize that a major problem with past development projects was that they were often designed unsystematically, leading to problems with reliability, maintainability, and extensibility.

To improve software development, the software-engineering discipline has tried to inject an engineering orientation into the design process and to encourage more standardization. This has resulted in a heavy emphasis on reuse of high-level designs and structural abstractions.

It is good software-engineering practice to reuse high-level abstractions, designs, logic, and structures. And what is good software-engineering practice should not be illegal. Copyright law should follow the lead of software engineering, not dictate to it what the parameters of this engineering discipline should be, particularly not by dogmatically assuming that whatever will promote progress in the literary arts will promote progress in the technological arts. Unfortunately, copyright law now treats software as a literary work, which the law affords the broadest protection, rather than as a factual work or a functional work, whose protection is narrower, as the box on p. 83 explains.

Also recall why copyright law has historically not protected such things as the designs in engineering drawings: because patent law already does so. Patent law has traditionally been used to regulate and promote technological progress. Unlike copyright law, which covers any original contribution no matter how minor, patent law does not give property rights to every original technological improvement that an engineer might design; it gives property rights only to significant advances in technology — to new inventions.

Granting property rights only to inventors is how patent law encourages people and firms to invest their resources in making significant improvements in technology. Incremental, trivial, or obvious improvements in technology will likely be

made without the extra boost that patent protection would provide. A fundamental principle of the patent system has been to leave more modest improvements in technology in the public domain, available for all to use as they want, so that incremental technological improvements may be widely available and competition among manufacturers might flourish.

And even for significant advances, patent law only grants property rights for 17 years, compared to a corporate copyright's 75 years, because 17 years of patent monopoly is long enough to provide proper incentives to technology creators without harming the public or slowing down the pace of innovation. Once a patent has expired, the invention is in the public domain and widens the available technological base from which engineers and other technologists can work.

If copyright law were to evolve as the Whelan case would have it, important public policies embodied in the patent system would be undermined. Trivial or obvious innovations in software design — things that would not qualify for any protection under patent law — would through copyright law become the private property of their creators for at least four times longer than if they had been patentable. Letting a firm have a monopoly on the logic, structure, and high-level designs for software systems for 75 years is too much — and an individual's copyright runs for life plus 50 years.

Why patents are better. One reason patent law is a superior body of law to protect software structure and high-level design is that it requires a creator to say exactly what his invention is. A patent for the organization of a piece of software would specify precisely what organizational elements the patent covers.

A copyright, by contrast, covers the undifferentiated whole of a protected work and unspecified aspects of it that courts may later construe to be expression. An author is under no obligation to specify exactly what in the copyrighted work he considers to be his property until he gets to court. Because of this, subsequent software developers can never be sure what previous copyright owners will claim to be their property.

Another reason patent law is superior to copyright is that it does not inhibit derivative innovations. To protect novelists from unauthorized screenplays of their works, copyright law provides that copyright owners have the exclusive right to control the preparation of derivative works, which are broadly defined as any work "based upon a preexisting work." By contrast, patent does not give inventors any rights over derivative works.

By permitting those who invent improvements on already patented technology to separately patent the improvements (even if they do not hold the underlying patent), patent law encourages derivative works to be made by others. And patent law lets you make modest improvements, such as modifications to make the technology more useful to the consumer who purchased it, without running the risk of liability to the patent holder.

Copyright law's much more hostile attitude toward derivatives means that if logic, structure, high-level design, and subroutine functions can be protected through copyright, the entire family of derivatives that these software features would be capable of would be significantly inhibited, if not totally stifled.

While some software-copyright owners might be willing to license others to use their high-level designs or structural abstractions, the point is that they need not do so. That's what having property rights means. While someone with a patent could deny a license to others, he could deny it for less time and could not control improvements others make to the patented work. Remember that if you offer an improvement over a copyrighted program, you give the originator an incentive to try to knock you out of the market through copyright litigation.

Even if the Whelan test is trimmed back so that the highest-level software design abstractions are not considered to be protectible expressions, there is a real danger that firms will be deterred by fear of copyright litigation from using what should be unprotectible high-level designs and structures because they can't be sure a court will be convinced how high the level of abstraction is. Even an independent-development defense may be difficult to establish because if you ever had access to the plaintiff's software and there are similarities between the products, that could overcome your independent-development defense.

Those who develop, refine, or distribute software need to be aware of developments in copyright law that affect software-engineering practices. One of the most contentious and unsettled issues in recent copyright cases is whether the structure of software is protectible expression under the copyright law. The Whelan decision says that it is, regardless of how high or low the level of abstraction may be.

Although the judges in that case thought their decision would promote innovation in the software field, they were wrong. Copyright protection for all structural elements of programs, regardless of their nature, will more likely retard than promote technological progress.

It is consistent with copyright principles, with good software-engineering practices, with common notions of fairness, and with the aim of advancing software as a technology to have copyright protect the instruction-by-instruction sequence of the software as well as the exact sequence of bits constituting the code. When another programmer has made only trivial variations on an existing program by changing variable names, by making slight switches in the sequence of instructions in a subroutine, or by recompiling the code — all of which cause two pieces of code to lose their identity — a court may still appropriately find copyright infringement.

But structural similarities at levels of abstraction above the instruction-by-instruction sequence probably should not be protected by copyright — a view consistent with copyright's rule of not protecting engineering designs. Strong structural protection under copyright law has too much potential to disrupt positive developments in the software-engineering community and to chill lawful uses of higher level abstractions by other developers.

What is needed is a way to give enough protection to stimulate investment in software innovation without impeding its

progress.

One approach is to modify copyright law and doctrine to achieve strong protection for software, but in this could stifle promising developments in software engineering that might make the US software industry more competitive in the global economy.

Another approach is to create a new form of intellectual-property protection for software, one similar to Congress's 1984 semiconductor-protection law, that would specify which features of software are protectible. The duration of protection should probably be considerably shorter than copyright's. (The semiconductor law, for example, provides only 10 years of protection.) Congress is not now considering any bills, but the drafters of the Semiconductor Protection Act thought it might be a precursor to a software-protection law.

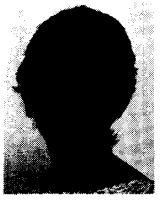
In 1986, the congressional Office of Technology Assessment also observed that software might need to be treated as a special case, like semiconductor designs. It proposed several alternatives to copyright because it concluded traditional copyright law was inappropriate for software.

A third approach is to reclassify software from a literary work to a functional work. This would be a step in the right direction, but it is not the whole answer because traditional copyright law does not protect truly functional works (like engineered products) and because the coexistent protection of copyright and patent still requires a test (a set of filters) to sort out what copyright covers, what patent covers, and what neither covers. In the past, no filter had been needed because the subjects that copyright and patent covered were mutually exclusive.

The fourth — and best — approach to protect innovative software designs and components is to use patent protection. This existing system seems to be the best vehicle for protecting software. ❖

Acknowledgment

I thank Robert Glushko for his most helpful editorial assistance with this article.



Pamela Samuelson is a professor of law at the University of Pittsburgh, where she teaches intellectual-property law. She is on sabbatical this year at Emory University. In 1985 and 1986, Samuelson was the principal investigator of the Software Engineering Institute's Software Licensing Project, which recommended several changes in the US Defense Dept.'s software-acquisition policy.

Samuelson received a Juris Doctor from Yale Law School and a BA in history from the University of Hawaii at Honolulu.

Address questions to the author at Emory University Law School, Gambrell Hall, Atlanta, GA 30322.