Against Software Patents

(February 28, 1991)

The League for Programming Freedom

Software patents threaten to devastate America's computer industry. Patents granted in the past decade are now being used to attack companies such as the Lotus Development Corporation for selling programs that they have independently developed. Soon new companies will often be barred from the software arena—most major programs will require licenses for dozens of patents, and this will make them infeasible. This problem has only one solution: software patents must be eliminated.

The Patent System and Computer Programs

The framers of the United States Constitution established the patent system so that inventors would have an incentive to share their inventions with the general public. In exchange for divulging an invention, the patent grants the inventor a 17 year monopoly on its use. The patent holder can license others to use the invention, but may also refuse to do so. Independent reinvention of the same technique by others does not give them the right to use it.

Patents do not cover specific systems: instead, they cover particular techniques that can be used to build systems, or particular features that systems can offer. Once a technique or feature is patented, it may not be used in a system without the permission of the patent-holder—even if it is implemented in a different way. Since a computer program typically uses many techniques and provides many features, it can infringe many patents at once.

Until recently, patents were not used in the software field. Software developers copyrighted individual programs or made them trade secrets. Copyright was traditionally understood to cover the implementation details of a particular program; it did not cover the features of the program, or the general methods used. And trade secrecy, by definition, could not prohibit any development work by someone who did not know the secret.

On this basis, software development was extremely profitable, and received considerable investment, without any prohibition on independent software development. But this scheme of things is no more. A change in U.S. government policy in the early 1980's stimulated a flood of applications. Now many have been approved, and the rate is accelerating.

Many programmers are unaware of the change and do not appreciate the magnitude of its effects. Today the lawsuits are just beginning.

Absurd Patents

The Patent Office and the courts have had a difficult time with computer software. 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. Patent examiners are often ill-prepared to evaluate software patent applications to determine if they represent techniques that are widely known or obvious—both of which are grounds for rejection.

Their task is made more difficult because many 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.

Computer scientists know many techniques that can be generalized to widely varying circumstances. But the Patent Office seems to believe that each separate use of a technique is a candidate for a new patent. For example, Apple was sued because the Hypercard program allegedly violates patent number 4,736,308, a patent that covers displaying portions of two or more strings together on the screen—effectively, scrolling with multiple subwindows. Scrolling and subwindows are well-known techniques, but combining them is now apparently illegal.

The granting of a patent by the Patent Office carries a presumption in law that the patent is valid. Patents for well-known techniques that were in use many years before the patent application have been upheld by federal courts. It can be hard to prove a technique was well known at the time in question.

For example, the technique of using exclusive-or to write a cursor onto a screen is both well known and obvious. (Its advantage is that another identical exclusive-or operation can be used to erase the cursor without damaging the other data on the screen.) This technique can be implemented in a few lines of a program, and a clever high school student might well reinvent it. But it is covered by patent number 4,197,590, which has been upheld twice in court even though the technique was used at least five years before the patent application. Cadtrak, the company that owns this patent, collects millions of dollars from large computer manufacturers.

English patents covering customary graphics techniques, including airbrushing, stenciling, and combination of two images under control of a third one, were recently upheld in court, despite the testimony of the pioneers of the field that they had developed these techniques years before. (The corresponding United States patents, including 4,633,416 and 4,602,286, have not yet been tested in court, but they probably will be soon.)

All the major developers of spreadsheet programs have been threatened on the basis of patent 4,398,249, covering "natural order recalc"—the recalculation of all the spreadsheet entries that are affected by the changes the user makes, rather than recalculation in a fixed order. Currently Lotus alone is being sued, but a victory for the plaintiff in this case would leave the other developers little hope. The League has found prior art that may defeat this patent, but this is not assured.

Nothing protects programmers from accidentally using a technique that is patented, and then being sued for it. Taking an existing program and making it run faster may also make it violate half a dozen patents that have been granted, or are about to be granted.

Even if the Patent Office learns to understand software better, the mistakes it is making now will follow us into the next century, unless Congress or the Supreme Court intervenes to declare these patents void.

However, this is not the whole of the problem. Computer programming is fundamentally different from the other fields that the patent system previously covered. Even if the patent system were to operate "as intended" for software, it would still obstruct the industry it is supposed to promote.

What Is “Obvious”?

The patent system will not grant or uphold patents that are judged to be obvious. However, the system interprets the word “obvious” in a way that might surprise computer programmers. The standard of obviousness developed in other fields is inappropriate for software.

Patent examiners and judges are accustomed to considering even small, incremental changes as deserving new patents. For example, the famous *Polaroid vs. Kodak* case hinged on differences in the number and order of layers of chemicals in a film—differences between the technique Kodak was using and those described by previous, expired patents. The court ruled that these differences were unobvious.

Computer scientists solve problems quickly because the medium of programming is tractable. They are trained to generalize solution principles from one problem to another. One such generalization is that a procedure can be repeated or subdivided. Programmers consider this obvious—but the Patent Office did not think that it was obvious when it granted the patent on scrolling multiple strings, described above.

Cases such as this cannot be considered errors. The patent system is functioning as it was designed to do—but with software, it produces outrageous results.

Patenting What Is Too Obvious to Publish

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.

For example, computer companies distributing the free X Window System developed by MIT are now being threatened with lawsuits by AT&T over patent number 4,555,775, covering the use of “backing store” in a window system that lets multiple programs have windows. Backing store means that the contents of a window that is temporarily partly hidden are saved in off-screen memory, so they can be restored quickly if the obscuring window disappears.

Early window systems were developed on computers that could not run two programs at once. These computers had small memories, so saving window contents was obviously a waste of scarce memory space. Later, larger multiprocessing computers led to the use of backing store, and to permitting each program to have its own windows. The combination was inevitable.

The technique of backing store was used at MIT in the Lisp Machine System before AT&T applied for a patent. (By coincidence, the Lisp Machine also supported multiprocessing.) The Lisp Machine developers published nothing about backing store at the time, considering it too obvious. It was mentioned when a programmers’ manual explained how to turn it on and off.

But this manual was published one week after the AT&T patent application—too late to count as prior art to defeat the patent. So the AT&T patent may stand, and MIT may be forbidden to continue using a method that MIT used before AT&T.

The result is that the dozens of companies and hundreds of thousands of users who accepted the software from MIT on the understanding that it was free are now faced with possible lawsuits. (They are also being threatened with Cadtrak’s exclusive-or patent.) The

X Window System project was intended to develop a window system that all developers could use freely. This public service goal seems to have been thwarted by patents.

Why Software Is Different

Software systems are much easier to design than hardware systems of the same number of components. For example, a program of 100,000 components might be 50,000 lines long and could be written by two good programmers in a year. The equipment needed for this costs less than $10,000; the only other cost would be the programmers' own living expenses while doing the job. The total investment would be less than a $100,000. If done commercially in a large company, it might cost twice that. By contrast, an automobile typically contains under 100,000 components; it requires a large team and costs tens of millions of dollars to design.

And software is also much cheaper to manufacture: copies can be made easily on an ordinary workstation costing under ten thousand dollars. To produce a complex hardware system often requires a factory costing tens of millions of dollars.

Why is this? A hardware system has to be designed using real components. They have varying costs; they have limits of operation; they may be sensitive to temperature, vibration or humidity; they may generate noise; they drain power; they may fail either momentarily or permanently. They must be physically assembled in their proper places, and they must be accessible for replacement in case they fail.

Moreover, each of the components in a hardware design is likely to affect the behavior of many others. This greatly complicates the task of determining what a hardware design will do: mathematical modeling may prove wrong when the design is built.

By contrast, a computer program is built out of ideal mathematical objects whose behavior is defined, not modeled approximately, by abstract rules. When an if-statement follows a while-statement, there is no need to study whether the if-statement will draw power from the while-statement and thereby distort its output, nor whether it could overstress the while-statement and make it fail.

Despite being built from simple parts, computer programs are incredibly complex. The program with 100,000 parts is as complex as an automobile, though far easier to design.

While programs cost substantially less to write, market and sell than automobiles, the cost of dealing with the patent system will not be less. The same number of components will, on the average, involve the same number techniques that might be patented.

The Danger of a Lawsuit

Under the current patent system, a software developer who wishes to follow the law must determine which patents a program violates and negotiate with each patent holder a license to use that patent. Licensing may be prohibitively expensive, or even unavailable if the patent is held by a competitor. Even "reasonable" license fees for several patents can add up to make a project infeasible. Alternatively, the developer may wish to avoid using the patent altogether; but there may be no way around it.

The worst danger of the patent system is that a developer might find, after releasing a product, that it infringes one or many patents. The resulting lawsuit and legal fees could force even a medium-size company out of business.

Worst of all, there is no practical way for a software developer to avoid this danger—there is no effective way to find out what patents a system will infringe. There is a way to try to find out—a patent search—but searches are unreliable and in any case too expensive to use for software projects.

Patent Searches Are Prohibitively Expensive

A system with a hundred thousand components can use hundreds of techniques that might already be patented. Since each patent search costs thousands of dollars, searching for all the possible points of danger could easily cost over a million. This is far more than the cost of writing the program.

The costs don't stop there. Patent applications are written by lawyers for lawyers. A programmer reading a patent may not believe that his program violates the patent, but a federal court may rule otherwise. It is thus now necessary to involve patent attorneys at every phase of program development.

Yet this only reduces the risk of being sued later—it does not eliminate the risk. So it is necessary to have a reserve of cash for the eventuality of a lawsuit.

When a company spends millions to design a hardware system, and plans to invest tens of millions to manufacture it, an extra million or two to pay for dealing with the patent system might be bearable. However, for the inexpensive programming project, the same extra cost is prohibitive. Individuals and small companies especially cannot afford these costs. Software patents will put an end to software entrepreneurs.

Patent Searches Are Unreliable

Even if developers could afford patent searches, these are not a reliable method of avoiding the use of patented techniques. This is because patent searches do not reveal pending patent applications (which are kept confidential by the Patent Office). Since it takes several years on the average for a software patent to be granted, this is a serious problem: a developer could begin designing a large program after a patent has been applied for, and release the program before the patent is approved. Only later will the developer learn that distribution of the program is prohibited.

For example, the implementors of the widely-used public domain data compression program `compress` followed an algorithm obtained from the journal *IEEE Computer*. (This algorithm is also used in several popular programs for microcomputers, including `PKZIP`.) They and the user community were surprised to learn later that patent number 4,558,302 had been issued to one of the authors of the article. Now Unisys is demanding royalties for using this algorithm. Although the program `compress` is still in the public domain, using it means risking a lawsuit.

The Patent Office does not have a workable scheme for classifying software patents. Patents are most frequently classified by end results, such as "converting iron to steel;" but many patents cover algorithms whose use in a program is entirely independent of the purpose of the program. For example, a program to analyze human speech might infringe the patent on a speedup in the Fast Fourier Transform; so might a program to perform symbolic algebra (in multiplying large numbers); but the category to search for such a patent would be hard to predict.

You might think it would be easy to keep a list of the patented software techniques, or even simply remember them. However, managing such a list is nearly impossible. A list compiled in 1989 by lawyers specializing in the field omitted some of the patents mentioned in this paper.

Obscure Patents

When you imagine an invention, you probably think of something that could be described in a few words, such as “a flying machine with fixed, curved wings” or “an electrical communicator with a microphone and a speaker”. But most patents cover complex detailed processes that have no simple descriptions—often they are speedups or variants of well-known processes that are themselves complex.

Most of these patents are neither obvious nor brilliant; they are obscure. A capable software designer will “invent” several such improvements in the course of a project. However, there are many avenues for improving a technique, so no single project is likely to find any given one.

For example, IBM has several patents (including patent number 4,656,583) on workmanlike, albeit complex, speedups for well-known computations performed by optimizing compilers, such as register coloring and computing the available expressions.

Patents are also granted on combinations of techniques that are already widely used. One example is IBM patent 4,742,450, which covers “shared copy-on-write segments.” This technique allows several programs to share the same piece of memory that represents information in a file; if any program writes a page in the file, that page is replaced by a copy in all of the programs, which continue to share that page with each other but no longer share with the file.

Shared segments and copy-on-write have been used since the 1960’s; this particular combination may be new as a specific feature, but is hardly an invention. Nevertheless, the Patent Office thought that it merited a patent, which must now be taken into account by the developer of any new operating system.

Obscure patents are like land mines: other developers are more likely to reinvent these techniques than to find out about the patents, and then they will be sued. The chance of running into any one of these patents is small, but they are so numerous that you cannot go far without hitting one. Every basic technique has many variations, and a small set of basic techniques can be combined in many ways. The patent office has now granted at least 2000 software patents—no less than 700 in 1989 alone, according to a list compiled by EDS. We can expect the pace to accelerate. In ten years, programmers will have no choice but to march on blindly and hope they are lucky.

Patent Licensing Has Problems, Too

Most large software companies are trying to solve the problem of patents by getting patents of their own. Then they hope to cross-license with the other large companies that own most of the patents, so they will be free to go on as before.

While this approach will allow companies like Microsoft, Apple and IBM to continue in business, it will shut new companies out of the field. A future start-up, with no patents of its own, will be forced to pay whatever price the giants choose to impose. That price might be high: established companies have an interest in excluding future competitors. The recent Lotus lawsuits against Borland and the Santa Cruz Operation (although involving an extended idea of copyright rather than patents) show how this can work.

Even the giants cannot protect themselves with cross-licensing from companies whose only business is to obtain exclusive rights to patents and then threaten to sue. For example, consider the New York-based Refac Technology Development Corporation, representing the owner of the "natural order recalc" patent. Contrary to its name, Refac does not develop anything except lawsuits—it has no business reason to join a cross-licensing compact. Cadtrak, the owner of the exclusive-or patent, is also a litigation company.

Refac is demanding five percent of sales of all major spread-sheet programs. If a future program infringes on twenty such patents—and this is not unlikely, given the complexity of computer programs and the broad applicability of many patents—the combined royalties could exceed 100% of the sales price. (In practice, just a few patents can make a program unprofitable.)

The Fundamental Question

According to the Constitution of the United States, the purpose of patents is to "promote the progress of science and the useful arts." Thus, the basic question at issue is whether software patents, supposedly a method of encouraging software progress, will truly do so, or will retard progress instead.

So far we have explained the ways in which patents will make ordinary software development difficult. But what of the intended benefits of patents: more invention, and more public disclosure of inventions? To what extent will these actually occur in the field of software?

There will be little benefit to society from software patents because invention in software was already flourishing before software patents, and inventions were normally published in journals for everyone to use. Invention flourished so strongly, in fact, that the same inventions were often found again and again.

In Software, Independent Reinvention Is Commonplace

A patent is an absolute monopoly; everyone is forbidden to use the patented process, even those who reinvent it independently. This policy implicitly assumes that inventions are rare and precious, since only in those circumstances is it beneficial.

The field of software is one of constant reinvention; as some people say, programmers throw away more "inventions" each week than other people develop in a year. And the comparative ease of designing large software systems makes it easy for many people to do work in the field. A programmer solves many problems in developing each program. These solutions are likely to be reinvented frequently as other programmers tackle similar problems.

The prevalence of independent reinvention negates the usual purpose of patents. Patents are intended to encourage inventions and, above all, the disclosure of inventions. If a technique will be reinvented frequently, there is no need to encourage more people to invent it; since some of the developers will choose to publish it (if publication is merited), there is no point in encouraging a particular inventor to publish it—not at the cost of inhibiting use of the technique.

Overemphasis of Inventions

Many analysts of American and Japanese industry have attributed Japanese success at producing quality products to the fact that they emphasize incremental improvements, convenient features and quality rather than noteworthy inventions.

It is especially true in software that success depends primarily on getting the details right. And that is most of the work in developing any useful software system. Inventions are a comparatively unimportant part of the job.

The idea of software patents is thus an example of the mistaken American preoccupation with inventions rather than products. And patents will encourage this mistaken focus, even as they impede the development work that actually produces better software.

Impeding Innovation

By reducing the number of programmers engaged in software development, software patents will actually impede innovation. 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.

When patents make development more difficult, and cut down on development projects, they will also cut down on the byproducts of development—new techniques.

Could Patents Ever Be Beneficial?

Although software patents in general are harmful to society as a whole, we do not claim that every single software patent is necessarily harmful. Careful study might show that under certain specific and narrow conditions (necessarily excluding the vast majority of cases) it is beneficial to grant software patents.

Nonetheless, the right thing to do now is to eliminate all software patents as soon as possible, before more damage is done. The careful study can come afterward.

Clearly software patents are not urgently needed by anyone except patent lawyers. The pre-patent software industry had no problem that was solved by patents; there was no shortage of invention, and no shortage of investment.

Complete elimination of software patents may not be the ideal solution, but it is close, and is a great improvement. Its very simplicity helps avoid a long delay while people argue about details.

If it is ever shown that software patents are beneficial in certain 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.

Software Patents Are Legally Questionable

It may come as a surprise that the extension of patent law to software is still legally questionable. It rests on an extreme interpretation of a particular 1981 Supreme Court decision, *Diamond vs. Deihr*.[1]

Traditionally, the only kinds of processes that could be patented were those for transforming matter (such as, for transforming iron into steel). Many other activities which we would consider processes were entirely excluded from patents, including business methods, data analysis, and "mental steps." This was called the "subject matter" doctrine.

Diamond vs. Deihr has been interpreted by the Patent Office as a reversal of this doctrine, but the court did not explicitly reject it. The case concerned a process for curing rubber—a transformation of matter. The issue at hand was whether the use of a computer program in the process was enough to render it unpatentable, and the court ruled that it was not. The Patent Office took this narrow decision as a green light for unlimited patenting of software techniques, and even for the use of software to perform specific well-known and customary activities.

Most patent lawyers have embraced the change, saying that the new boundaries of patents should be defined over decades by a series of expensive court cases. Such a course of action will certainly be good for patent lawyers, but it is unlikely to be good for software developers and users.

One Way to Eliminate Software Patents

We recommend the passage of a law to exclude software from the domain of patents. That is to say that, no matter what patents might exist, they would not cover implementations in software; only implementations in the form of hard-to-design hardware would be covered. An advantage of this method is that it would not be necessary to classify patent applications into hardware and software when examining them.

Many have asked how to define software for this purpose—where the line should be drawn. For the purpose of this legislation, software should be defined by the characteristics that make software patents especially harmful:

- Software is built from ideal infallible mathematical components, whose outputs are not affected by the components they feed into.

  Ideal mathematical components are defined by abstract rules, so that failure of a component is by definition impossible. The behavior of any system built of these components is likewise defined by the consequences of applying the rules step by step to the components.

[1] See "Legally Speaking" in *Communications of the ACM*, August 1990.

- Software can be easily and cheaply copied.

Following this criterion, a program to compute prime numbers is a piece of software. A mechanical device designed specifically to perform the same computation is not software, since mechanical components have friction, can interfere with each other's motion, can fail, and must be assembled physically to form a working machine.

Any piece of software needs a hardware platform in order to run. The software operates the features of the hardware in some combination, under a plan. Our proposal is that combining the features in this way can never create infringement. If the hardware alone does not infringe a patent, then using it in a particular fashion under control of a program should not infringe either. In effect, a program is an extension of the programmer's mind, acting as a proxy for the programmer to control the hardware.

Usually the hardware is a general purpose computer, which implies no particular application. Such hardware cannot infringe any patents except those covering the construction of computers. Our proposal means that, when a user runs such a program on a general purpose computer, no patents other than those should apply.

The traditional distinction between hardware and software involves a complex of characteristics that used to go hand in hand. Some newer technologies, such as gate arrays and silicon compilers, blur the distinction because they combine characteristics associated with hardware with others associated with software. However, most of these technologies can be classified unambiguously for patent purposes, either as software or as hardware, using the criteria above. A few gray areas may remain, but these are comparatively small, and need not be an obstacle to solving the problems patents pose for ordinary software development. They will eventually be treated as hardware, as software, or as something in between.

What You Can Do

One way to help eliminate software patents is to join the League for Programming Freedom. The League is a grass-roots organization of programmers and users opposing software patents and interface copyrights. (The League is not opposed to copyright on individual programs.) Annual dues for individual members are \$42 for employed professionals, \$10.50 for students, and \$21 for others. We appreciate activists, but members who cannot contribute their time are also welcome.

To contact the League, phone (617) 243-4091, send Internet mail to the address `lpf@uunet.uu.net`, or write to:

League for Programming Freedom
1 Kendall Square #143
PO Box 9171
Cambridge, MA 02139

In the United States, another way to help is to write to Congress. You can write to your own representatives, but it may be even more effective to write to the subcommittees that consider such issues:

House Subcommittee on Intellectual Property
2137 Rayburn Bldg
Washington, DC 20515

Senate Subcommittee on Patents, Trademarks and Copyrights
United States Senate
Washington, DC 20510

You can phones your representatives at (202) 225-3121, or write to them using the following addresses:

Senator So and So
United States Senate
Washington, DC 20510

Representative Such and Such
House of Representatives
Washington, DC 20515

Fighting Patents One by One

Until we succeed in eliminating all patenting of software, we must try to overturn individual software patents. This is very expensive and can solve only a small part of the problem, but that is better than nothing.

Overturning patents in court requires prior art, which may not be easy to find. The League for Programming Freedom will try to serve as a clearing house for this information, to assist the defendants in software patent suits. This depends on your help. If you know about prior art for any software patent, please send the information to the League at the address given above.

If you work on software, you can personally help prevent software patents by refusing to cooperate in applying for them. The details of this may depend on the situation.

Conclusion

Exempting software from the scope of patents will protect software developers from the insupportable cost of patent searches, the wasteful struggle to find a way clear of known patents, and the unavoidable danger of lawsuits.

If nothing is changed, what is now an efficient creative activity will become prohibitively expensive. To picture the effects, imagine if each square of pavement on the sidewalk had an owner, and pedestrians required a license to step on it. Imagine the negotiations necessary to walk an entire block under this system. That is what writing a program will be like if software patents continue. The sparks of creativity and individualism that have driven the computer revolution will be snuffed out.