# Against Software Patents

(draft)

The League for Programming Freedom

Software patents threaten to devastate America's computer industry. Newly-granted software patents are being used to attack companies such as the Lotus Development Corporation and Microsoft for selling programs that they have independently developed. Soon new companies may be barred from entering the software arena, because the cost of licensing the dozens of patents necessary for a major program will make such a project economically impossible.

As programmers, we believe that if the United States Patent and Trademark Office continues to grant software patents, we will soon be effectively forbidden from writing programs that are useful.

## The Patent System And Computer Programs

The framers of the 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 the use of the invention. 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 let them use it.

Until recently, patents were simply not used in the field of software. Software developers would copyright individual programs, or make them trade secrets.

Copyright was traditionally understood to cover the particular 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 prohibiting the development of new programs by others.

But this scheme of things is no more. Software patents became legal in the U.S. in 1981, and now enough time has elapsed for numerous patents to be approved.

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 very difficult time with computer software. The Patent Office refuses 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 which have been previously used or are 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, others seemed insufficiently general. Complicated assemblages of techniques have often been kept secret.

And what is obvious to a programmer is frequently not obvious to a patent examiner, many of whom view innovations in computer science the same way as they see innovations in chemistry or biology. Computer scientists know many techniques that can be generalized to widely varying circumstances. Based on patents that have been awarded, the Patent Office seems to believe that each separate use of a technique is a candidate for a patent.

For example, Apple has been sued because the Hypercard program violates patent number 4,736,308, a patent that describes nested scrollable objects: windows that can scroll, containing tables that can individually scroll, containing items that can individually scroll. These three types of scrolling were all in use at the time that patent number 4,736,308 was applied for, but combining them is now illegal.

Many well-known and widely used techniques have been patented. Unfortunately, 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 for more than 10 years before the patent was granted have been upheld by federal courts.

For example, the technique of using exclusive-or to write a cursor onto a screen is well known, and has been used for decades. (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 used in just a few lines of program, and a clever high school student might well reinvent it. But this, as well as other important graphics techniques, is covered by patent number 4,197,590, which has been upheld twice in court.

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.)

Currently all companies who have developed spreadsheet programs are being sued because of a 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. This technique is very similar to the old artificial intelligence techniques of antecedent reasoning and constraint propagation, but we cannot rely on the courts to overturn the patent on these grounds.

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 them void.

However, this is not the extent of the problem. Computer programming is fundamentally different from the other fields that the patent system previously covered. As a result, even if the patent system were fixed to operate "as intended" for software, it would still largely wipe out the industry it is ostensibly designed to encourage.

## 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 a hundred thousand components might be fifty thousand lines long and could be written by two good programmers in a year. The equipment needed for this costs less than ten thousand dollars; the only other cost would be the programmers' own living expenses while doing the job. The total investment would be less than a hundred thousand dollars. If done commercially in a large company, it might cost twice that. By contrast, an automobile typically contains under a hundred thousand 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 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 inserted in their place in the machinery, and it must be possible to gain access to them to test or replace them.

Moreover, each of the components in a hardware design is likely to affect the behavior of many others. Therefore, is it very hard to figure out 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 merely modeled approximately, by abstract rules. When you write an if-statement after a while-statement, you don't have to worry that the if-statement will draw power from the while-statement and thereby distort its output, nor that it will overstress the while-statement and make it fail.

Despite the fact that they are built from simple parts, computer programs are incredibly complex. The program with fifty thousand lines probably has a hundred thousand parts, making it 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 is not less. The same number of components will, in general, be likely to involve the same number of possibly-patented techniques.

## What Is "Obvious"?

The patent system will not grant or uphold patents that are judged to be "obvious." However, the standard of obviousness that the patent system has developed in other fields is inappropriate to the software field.

Patent examiners are accustomed to considering even small, incremental changes as deserving new patents. For example, the famous *Polaroid vs. Kodak* case turned 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 far faster than people in other disciplines, because the medium of programming is more tractable. So they are trained to generalize solution

principles from one problem to another. One such generalization is that a procedure can be repeated within itself, a process known as nesting. Nesting in software is obvious to computer programmers—but the Patent Office did not think that it was obvious when it granted the patent on nested scrolling, for which Apple was sued.

Cases such as this cannot be considered errors. The patent system is functioning in software just as it does in other fields—but with software, the result is outrageous.

## The Danger of a Lawsuit

Under the current patent system, a software developer who wishes to follow the law must determine which patents his program violates and negotiate with each patent holder a license to use that patent. Licensing may be prohibitively expensive, as in the case when the patent is held by a competitor. Even "reasonable" license fees for several patents can add up to make a project unfeasible. Alternatively, the developer may wish to avoid using the patent altogether; unfortunately, 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 such searches are unreliable and in any case too expensive to use for software projects.

## Patent Searches Are Prohibitively Expensive

In a system with a hundred thousand components, there can easily be 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.

But 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 such involvement 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.

In particular, individuals and small companies cannot afford these costs. Software patents will put an end to software entrepreneurs.

## Patent Searches Are Unreliable

Even if companies could afford the heavy cost of patent searches, they 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 patent to be granted, this is a serious problem: a company 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 that company find out whether its profits will be confiscated.

For example, the implementors of the widely-used public domain program `compress` followed an algorithm obtained from the *Communications of the ACM*. Only later, when the program was in widespread use, did the community learn that the authors had received patent number 4,558,302. Now Unisys is demanding royalties for using this algorithm. Although the program is still in the public domain, no one is legally allowed to use it any more. And using what you learned >from the journals is no longer safe.

In addition, the Patent Office does not have a workable scheme for classifying software patents. Patents are most frequently classified by the activity they are used in, 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 in practice. The patent office has now granted more than 2000 software patents. In 1989 alone, 700 patents were issued. We can expect the pace to accelerate.

When you think of inventions, you probably call to mind revolutionary inventions such as the telephone or magnetic core memory. This is not the standard that the patent system uses, however. What we would consider a minor cleverness or variation or combination of existing techniques, they consider patentable. This leads to a profusion of obscure patents.

Any capable software designer will "invent" several such improvements in the course of a project, and will say that they are straightforward—hardly inventions at all. However, the number of avenues for such improvement is very large, so no single project is likely to find any given one. Therefore, the Patent Office is not likely to classify them as obvious. As a result, IBM has several patents (including 4,656,583) on certain fairly straightforward, albeit complex, speedups for well-known computations performed by optimizing compilers, such as computing the available expressions and register coloring.

Patents are also granted on combinations of techniques that are already well known and in use. One example is IBM patent 4,742,450, which covers "shared copy-on-write segments." This is a technique that 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 are very old techniques; this particular combination may be new as an advertised 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.

These sorts of patents are like land mines: your chances of running into any one of them are small, but soon there will be thousands of them. Even today it is hard to keep track of them, and a recent list published by lawyers specializing in the field omitted some of these IBM patents. 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 all the other companies and be free to go on as before.

While this approach will allow companies like Microsoft, Apple and IBM to continue business, it will shut future companies out of the marketplace. A future start-up, with no patents of its own, will have no choice but to meet whatever conditions the giants choose to impose. And that price might be extremely high: companies currently in the market have an incentive to keep out 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 a system of industry-wide cross-licensing will not protect the software industry from companies whose only business is to buy patents and then sue people for license fees. For example, the New York-based REFAC Technology Development Corporation recently bought the rights to the "natural order recalc" patent, solely so that REFAC could sue Lotus, Microsoft and other companies selling spread-sheet programs. Contrary to its name, REFAC does not develop anything except lawsuits. It has no financial incentive to join a cross-licensing compact. The exclusive-or patent is owned by another such litigation company, Cadtrak, which is now suing Western Digital.

REFAC is demanding five percent of sales of all major spread-sheet programs. If some future program infringes on twenty such patents—and this is not at all unlikely, given the complexity of a computer program and the specificity of patents that have been recently issued—that program will never be used.

To get a picture of the effects for yourself, imagine if each square of pavement on the sidewalk had its owner, and you had to negotiate a license to step on it. Imagine trying to walk the entire length of a block under this system. That is what writing a program will be like if software patents are allowed to proliferate.

## 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 whether they will instead hold progress back.

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; anyone who uses the patented technique can be stopped, even if it was independently reinvented.

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.

As programmers, we solve many problems each time we develop a program. In the past, we would publish the important solutions in journals, and forget the rest. All of these solutions are likely to be reinvented frequently as additional people tackle similar problems and try to do a good job.

Today, however, many of these specialized solutions are being patented. If you then rediscover it in the course of your work, you are headed for a lawsuit that you cannot anticipate.

Meanwhile, the prevalence of independent reinvention negates the usual justification for patents. Patents are intended to encourage the development of 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 it merits publication), there is no point in encouraging a particular inventor to do so—and certainly not at such a high price.

## Could Patents Ever Be Beneficial?

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

This 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.

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

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.

## Inventions Are Not the Important Thing

Many observers of US and Japanese industry have noted that one of the reasons Japanese are better at producing quality products is that they assign greater importance to incremental improvements, convenient features and quality rather than to 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 small part of the process.

The idea of software patents is thus an example of the mistaken American preoccupation with the big invention rather than the desirable product. Patents will reinforce this misdirection of American attention. Meanwhile, by presenting obstacles to competition in the important part of software development, they will interfere with development of quality software.

## 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 did 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 what can be patented should be defined over decades by a series of expensive court cases. Such a course of action will certainly be good for the patent lawyers, but it is unlikely to be good for software developers and users.

---

[1] This information comes from a paper being written by Professor Samuelson of the Emory School of Law.

## One Way to Eliminate Software Patents

We recommend that Congress pass a law that excludes software from the domain of patents. That is to say that, no matter what might be patented, the patent 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 judging them.

People often ask how it would be possible to define software for this purpose—where the line would be drawn.

For the purpose of this legislation, software should be defined by precisely the characteristics that make software patents harmful:

- Software is built from ideal mathematical components, whose inputs are clearly distinguished from their outputs.

  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 to its components.
- Software can be easily and cheaply copied.

Thus, a program which computes prime numbers is a piece of software. A mechanical device designed specifically to perform the same computation would not be software, since the mechanical device might fail if it were not properly oiled, and would have to be constructed out of physical objects.

There are areas of design which are between hardware and software in some ways: for example, gate arrays and silicon compilers. These will fall on one side or the other of the line that is drawn. If the line is drawn as proposed here, based on the needs of the field, there is reason to hope that these will fall on the side that is best. However, these in-between areas are comparatively small, and what really matters is to solve the problem for the larger area of ordinary software as surely and expeditiously as possible.

## What You Can Do

One way to help oppose software patents is to join the League for Programming Freedom. The League is a grass-roots organization of programmers and users dedicated to preserving the freedom to develop software that does what users want. The League opposes software patents and user interface copyrights, and advocates a return to the legal system for software that existed a decade ago.

Annual dues for individual members are $42 for employed professionals, $10.50 for students, and $21 for others. We appreciate activists, but members who have no free time to contribute are also welcome.

You can phone (617) 243-4091, send electronic mail to `league@prep.ai.mit.edu`, 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 write to your own representatives using the following addresses:

Senator So and So
United States Senate
Washington, DC 20510

Representative Such and Such
House of Representatives
Washington, DC 20515

You can phone senators and representatives at (202) 225-3121.

## Conclusion

Exempting software from the scope of patents will prevent the patent system from turning an efficient creative activity into something that is prohibitively expensive. Individual practitioners will be able to continue work in their fields without expensive patent searches, the struggle to find a way clear of patents, and the unavoidable danger of lawsuits.

If this change is not made, it is quite possible that the sparks of creativity and individualism that have driven the computer revolution will be snuffed out.