SIMSON L. GARFINKEL
RICHARD M. STALLMAN
MITCHELL KAPOR

Why Patents Are Bad for Software[1]

Patents can't protect or invigorate the computer software industry; they can only cripple it.

In September 1990, users of the popular XyWrite word processing program got a disturbing letter in the mail from XyQuest, Inc., the program's publisher:

> In June of 1987, we introduced an automatic correction and abbreviation expansion feature in XyWrite III Plus. Unbeknownst to us, a patent application for a related capability had been filed in 1984 and was subsequently granted in 1988. The company holding the patent contacted us in late 1989 and apprised us of the existence of their patent.
>
> We have decided to modify XyWrite III Plus so that it cannot be construed as infringing. The newest version of XyWrite III Plus (3.56) incorporates two significant changes that address this issue: You will no longer be able to automatically correct common spelling errors by pressing the space bar after the misspelled word. In addition, to expand abbreviations stored in your personal dictionary, you will have to press control-R or another designated hot key.

XyQuest had been bitten by a software patent—one of the more than two thousand patents on computer algorithms and software techniques that have been granted by the U.S. Patent and Trademark Office since the mid-1980s. The owner of the patent, Productivity Software, had given XyQuest a choice: license the patent or take a popular feature out of XyWrite, XyQuest's flagship product. If XyQuest refused, a costly patent-infringement lawsuit was sure to follow.

Some choice.

XyQuest tried to license the patent, says Jim Adelson, vice president for marketing, but Productivity Software kept changing its terms. First Productivity said that XyQuest could keep the feature in some versions of XyWrite, but not in others. Then the company said that XyQuest could use one part of the "invention," but not other parts. And Productivity Software kept increasing the amount of money it wanted. XyQuest finally gave up and took the feature out.

XyQuest was lucky it had that option. Other firms—including some of the nation's largest and most profitable software publishers—have been served with notice of patents that strike to the heart of their corporate vitality. In one of the most publicized cases, a company called Refac International—whose sole business is acquiring and litigating patents—sued Lotus, Microsoft, Ashton-Tate, and three other spreadsheet publishers, claiming they had all infringed on patent number 4,398,249, which spells out the order in which to recalculate the values in a complicated model when one parameter in the model changes. (Refac has since dropped its claims against all the companies except Lotus, but only because company lawyers anticipated a better chance of success if they faced just one opponent.)

Patent 4,398,249 does not have anything to do with spreadsheets in particular; the technique also appears in some graphics drawing and artificial intelligence programs. And the idea that

[1] From Issues in Science and Technology, Fall 1991.

values in a spreadsheet should be recalculated in the order specified by the patent is so obvious that it has probably occurred to nearly everyone who has written a spreadsheet program. But the Patent Office's standard for obviousness is extremely low; patents have been granted for ideas so elementary that they could have been answers to problems in a first-year programming course.

Practically once a month, the nation's computer networks are abuzz with news of another patent issued on a fundamental concept that is widely used. Although the Patent Office isn't supposed to grant patents on ideas, that's essentially what it's doing with software patents, carving up the intellectual domain of computer science and handing little pieces to virtually any company that files an application. And the practice is devastating America's software industry.

If Congress does not act quickly to redefine the applicability of patent law to computer programs, the legal minefield confronting the introduction of new computer programs will be so intimidating—and potentially so costly—that small companies will effectively be barred from the marketplace, while large, established firms will become embroiled in litigation that will have a stultifying effect on the entire industry.

1 What's being patented?

Software patents do not cover entire programs; instead, they cover algorithms and techniques—the instructions that tell a computer how to carry out a specific task in a program. Thousands of instructions make up any one computer program. But whereas the unique combination of algorithms and techniques in a program is considered an "expression" (like a book or a song) and is covered by copyright law, the algorithms and techniques themselves are treated as procedures eligible for patenting.

The judicial basis for this eligibility is tenuous at best. U.S. law does not allow inventors, no matter how brilliant they are, to patent the laws of nature, and in two Supreme Court cases (Gottschalk v. Benson, 1972, and Parker v. Flook, 1978) the Court extended this principle to computer algorithms and software techniques. But in the 1981 case Diamond v. Diehr, the Court said that a patent could be granted for an industrial process that was controlled by certain computer algorithms, and the Patent Office seems to have taken that decision as a green light on the patentability of algorithms and techniques in general.

Software patents are now being granted at an alarming rate—by some counts, more than a thousand are issued each year. Unfortunately, most of the patents have about as much cleverness and originality as a recipe for boiled rice—simple in itself but a vital part of many sophisticated dishes. Many cover very small and specific algorithms or techniques that are used in a wide variety of programs. Frequently the "inventions" mentioned in a patent application have been independently formulated and are already in use by other programmers when the application is filed.

When the Patent Office grants a patent on an algorithm or technique, it is telling programmers that they may not use a particular method for solving a problem without the permission of the idea's "owner." To them, patenting an algorithm or technique is like patenting a series of musical notes or a chord progression, then forcing composers to purchase a "musical sequence license."

2 Systems at odds

The traditional rationale for patents is that protection of inventions will spur innovation and aid in the dissemination of information about technical advances. By prohibiting others from copying an invention, patents allow inventors to recoup their investment in development while at the same time revealing the workings of the new invention to the public.

But there's evidence that the patent system is backfiring in the computer industry; indeed, the system itself seems unsuited to the nature of software development. Today's computer programs are so complex that they contain literally thousands of algorithms and techniques, each considered patentable by the Patent Office's standards. Is it reasonable to expect a software company to license each of those patents, or even to bring such a legally risky product into the marketplace? To make things even more complicated, the Patent Office has also granted patents on combinations of algorithms and techniques that produce a particular feature. For example, Apple was sued because its Hypercard program allegedly violates patent number 4,736,308, which covers a specific technique that, in simplified terms, entails scrolling through a database displaying selected parts of each line of text. Separately, the scrolling and display functions are ubiquitous fixtures of computer programming, but combining them without a license from the holder of patent 4,736,308 is now apparently illegal.

Another problem with patenting software is the amount of time it takes to do so. The two to five years required to file for and obtain a patent are acceptable if a company is patenting, say, the formula for Valium, which hasn't changed in more than 20 years. But in the software industry, companies that don't continually bring out new versions of their programs go out of business. Success for them depends on spotting needs and developing solutions as quickly as possible.

Unfortunately, conducting a patent search is a slow, deliberative process that, when harnessed to software development, could stop innovation in its tracks. And because patent applications are confidential, there is simply no way for computer programmers to ensure that what they write will not violate some patent that is yet to be issued. Thus XyQuest "reinvented" its automatic spelling-error correction system and brought the product to market between the time that Productivity Software had filed for its application and been awarded the patent.

Such examples are becoming increasingly common. In another case, the journal IEEE Computer in June 1984 published a highly efficient algorithm for performing data compression; unbeknownst to the journal's editors or readers, the authors of the article had simultaneously applied for a patent on their invention. In the following year, numerous programs were written and widely distributed for performing the so-called "LZW data compression." The compression system was even adopted as a national standard and proposed as an international one. Then, in 1985, the Patent Office awarded patent number 4,558,302 to one of the authors of the article. Now Unisys, the holder of the patent, is demanding royalties for the use of the algorithm. Although programs incorporating the algorithm are still in the public domain, using these programs means risking a lawsuit.

Not only is the patent approval process slow, but the search for "prior art"—the criterion the Patent Office uses to determine whether an invention already exists at the time of a patent application—is all but impossible to conduct in the realm of computer software. After more than 25 years, the Patent Office has not developed a system for classifying patents on algorithms and techniques, and no such system may be workable. Just as mathematicians are sometimes unaware that essentially identical mental processes are being used in separate areas of mathematics under different terminology, different parts of computer science frequently reinvent the same algorithm to serve different purposes. It is unreasonable to expect that a patent examiner, pressed for time, would recognize all such duplication. For example, IBM was issued a patent on the same data-compression algorithm that Unisys supposedly owns. The Patent Office was probably not aware of granting two patents for the same algorithm because the descriptions in the patents themselves are quite different even though the formulas are mathematically equivalent.

The search for prior art is complicated by the fact that the literature of computer science is unbelievably large. It contains not only academic journals, but also users' manuals, published source code, and popular accounts in magazines for computer enthusiasts. Whereas a team of chemists working at a major university might produce 20 or 30 pages of published material per year, a single programmer might easily produce a hundred times that much. The situation becomes even more complex in the case of patented combinations of algorithms and techniques. Programmers often publish new algorithms and techniques, but they almost never publish new ways of combining old ones. Although individual algorithms and techniques have been combined in many different ways in the past, there's no good way to establish that history.

The inability to search the literature thoroughly for prior art is crucial, because unless an examiner can find prior art, he or she is all but obligated to issue the patent. As a result, many patents have been granted—and successfully defended in court—that are not "original," even by the Patent Office's definition. It was simply the case that neither the patent examiner nor the defendants in the lawsuit knew of the prior art's existence.

Some members of the commercial software community are now proposing the creation of a "Software Patent Institute" to identify software's prior art that existed before 1980. But even if such an institute could catalogue every discovery made by every programmer in the United States, it makes no sense to arbitrarily declare that only pre-1980 work is in the public domain. Besides, what would be the purpose? To allow the patenting of nature's mathematical laws?

3 Bad for business

Even when patents are known in advance, software publishers have generally not licensed the algorithms or techniques; instead, they try to rewrite their programs to avoid using the particular procedure that the patent describes. Sometimes this isn't possible, in which case companies have often chosen to avoid implementing new features altogether. It seems clear from the evidence of the last few years that software patents are actually preventing the adoption of new technology, rather than encouraging it.

And they don't seem to be encouraging innovation, either. Software patents pose a special danger to small companies, which often form the vanguard of software development but can't afford the cost of patent searches or litigation. The programming of a new product can cost a few hundred thousand dollars; the cost of a patent search for each technique and combination of techniques that the new program uses could easily equal or even exceed that. And the cost of a single patent suit can be more than a million dollars.

"I'm not familiar with any type of ligation that is any more costly than patent litigation," says R. Duff Thompson, vice president and general counsel of the WordPerfect Corporation. But Thompson's greatest fear is that software patents will wipe out young, independent programmers, who until now have been the software industry's source of inspiration. Imagine what happens, says Thompson, when "some 23-year-old kid who has a terrific idea in a piece of software is hammered by a demand letter from someone holding a patent."

As for aiding the exchange of information, the expansion of software patents could mean instead the end of software developed at universities and distributed without charge—software that has been a mainstay of computer users in universities, corporations, and government for years. Many such programs—the X Window system, the Emacs text editor, the "compress" file-compression utility, and others—appear to be in violation of existing patents. Patents could also mean an end to public-domain software, which has played an important part in making computers affordable to public schools. There is obviously no way that an author who distributes a program for free could arrange to pay for royalties if one of the hundreds of techniques that were combined to create the program happens to be patented.

Few programmers and entrepreneurs believe that patents are necessary for their profession. Instead, the impetus for patents on algorithms and techniques comes from two outside sources: managers of large companies, who see patents as a means for triumphing over their competitors without having to develop superior products, and patent attorneys, who see the potential for greatly expanding their business.

Today, most patenting by companies is done to have something to trade or as a defense against other patent-infringement suits. Attorneys advise that patenting software may strengthen competitive position. Although this approach will work for large companies such as Microsoft, Apple, and IBM, small and even mid-sized companies can't play in their league. A future startup will be forced to pay whatever price the giants choose to impose.

4 Copyright and trade secrecy

The best argument against the wisdom of software patents may be history itself. Lotus, Microsoft, WordPerfect, and Novell all became world leaders in the software publishing industry on the strength of their products. None of these companies needed patents to secure funding or maintain their market position. Indeed, all made their fortunes before the current explosion of software patents began. Clearly patents are not necessary to ensure the development of computer programs. And for those who want more control over what they see as their property, the computer industry has already adopted two other systems: copyright and trade secrecy.

Today, nearly all programs are copyrighted. Copyright prohibits the users of a software program from making copies of it (for example, to give to their friends) without the permission of the individual or company that licenses the program. It prevents one company from appropriating another company's work and selling it as its own. But the existence of a copyright doesn't prevent other programmers from using algorithms or techniques contained in the program in their own work. A single software technique can be implemented in different ways to do totally different jobs; copyright only prohibits appropriating the actual code that a particular programmer wrote.

In general, copyrighting and patenting are thought to apply to very different kinds of material: the former to the expression of ideas, and the latter to a process that achieves a certain result. Until just a few years ago, computer algorithms and techniques were widely seen as unpatentable. And as Harvard University policy analyst Brian Kahin notes, this is the first time in history that an industry in which copyright was widely established was suddenly subjected to patenting.

Indeed, without conscious action by Congress or the Supreme Court, the most fundamental rule of software publishing—if you write a program, you own it—will change. The new rule will be that you might own what you write—if it is so revolutionary that it owes nothing to any previous work. No author in areas other than software is held to such an unrealistically high standard.

The U.S. patent system was created because the framers of the Constitution hoped that patents would discourage trade secrecy. When techniques are kept secret for commercial advantage, they may never become available for others to use and may even be lost. But although trade secrecy is a problem for software, as it is for other fields, it is not a problem that patents help to correct.

Many of the useful developments in the field of software consist of new features such as the automatic correction and abbreviation expansion feature in XyWrite III Plus. Since it is impossible to keep a program's features secret from the users of the program, there is no possibility of trade secrecy and thus no need for measures to discourage it. Techniques used internally in a software system can be kept secret; but in the past, the important ones rarely were. It was normal for computer scientists in the commercial as well as the academic world to publish their discoveries. Once again, since secrecy about techniques was not a significant problem, there is little to be gained by adopting the patent system to discourage it.

The place where trade secrecy is used extensively in software is in the "source code" for programs. In computer programming, trade secrets are kept by distributing programs in "machine code," the virtually indecipherable translation of programming languages that computers read. It is extremely difficult for another programmer to glean from a machine-code program the original steps written by the program's author. But software patents haven't done anything to limit this form of trade secrecy. By withholding the source code, companies keep secret not a particular technique, but the way that they have combined dozens of techniques to produce a design for a complete system. Patenting the whole design is impractical and ineffective. Even companies that have software patents still distribute programs in machine code only. Thus, in no area do software patents significantly reduce trade secrecy.

5 Reversing direction

Many policymakers assume that any increase in intellectual property protection must be good for whoever works in the field. As we've tried to show, this is assuredly not the case in the field of computer programming. Nearly all programmers view patents as an unwelcome intrusion, limiting both their ability to do their work and their freedom of expression.

At this point, so many patents have been issued by the Patent and Trademark Office that the prospect of overturning them by finding prior art, one at a time, is almost unthinkable. Even if the Patent Office learns to understand software better in the future, the mistakes that are being made now will follow the industry into the next century unless there is a dramatic turnaround in policy.

The U.S. Patent and Trademark Office recently established an Advisory Commission on Patent Law Reform that is charged with examining a number of issues, including software patents—or what it prefers to call patents on "computer-program-related inventions." Unfortunately, the commission's subcommittee on software does not include any prominent software industry representatives who have expressed doubts about software patents. But the subcommittee is required to consider public comment. The commission's final report is not due until August 1992, so there is still time to make one's voice heard.

Although influencing the Patent Office might produce some benefits, the really necessary reforms are likely to come only through intervention by the Supreme Court or Congress. Waiting for Court action is not the answer: No one can force the Supreme Court to rule on a relevant case, and there is no guarantee that the Court would decide to change Patent Office practice or to do anything about existing patents. The most effective course of action, therefore, is to encourage Congress to amend the patent law to disallow software patents and, if possible, invalidate those that have already been awarded. The House Subcommittee on Intellectual Property and the Administration of Justice, chaired by Representative William J. Hughes (D-N.J.), should take the lead by scheduling hearings on the subject and calling for a congressionally sponsored economic analysis of the effect of software patents on the industry.

The computer industry grew to be vibrant and healthy without patents. Unless those who want software patents can demonstrate that they are necessary to the health of the industry, Congress should feel justified in eliminating this barrier to innovation.

6 Recommended reading

- Brian Kahin, "The Software Patent Crisis," *Technology Review* (April 1990): 53-58.
- Mitchell Kapor, Testimony at Hearings before U.S. House of Representatives, Subcommittee on Courts, Intellectual Property and the Administration of Justice, of the Committee on the Judiciary (March 5, 1990).
- Pamela Samuelson, "Benson Revisited: Should Patent Protection Be Available for Algorithms and Other Computer Program-Related Inventions?" *Emory Law Journal* (Fall 1990): 1025-1154.
- Pamela Samuelson, "Should Program Algorithms Be Patented?" *Communications of the ACM* (August 1990): 23-27.

Simson Garfinkel is a senior editor at *NeXTWORLD* magazine (San Francisco) and the coauthor of the book *Practical UNIX Security*.

Richard M. Stallman is one of the founders of the League for Programming Freedom (Cambridge, Massachusetts) and the recipient of the Association for Computing Machinery's Grace Hopper Award.

Mitchell Kapor, a founder of the Lotus Development Corporation, is president of the Electronic Frontier Foundation (Cambridge, Massachusetts).