OPTIMA

№ 33

MATHEMATICAL PROGRAMMING SOCIETY NEWSLETTER *June 1991*

Special Edition

Report of the Committee ON Algorithms and the Law

George B. Dantzig

Donald Goldfarb

Eugene Lawler

Clyde Monma

Stephen M. Robinson
(CHAIR)

MATHEMATICAL PROGRAMMING SOCIETY

OPTIMA
NUMBER 33

O P T I M A

Background and charge

THE Committee was appointed in the spring of 1990 by George Nemhauser, Chairman of the Mathematical Programming Society (MPS). Its charge follows:

"The purpose of the committee should be to devise a position for MPS to adopt and publicize regarding the effects of patents on the advancement of research and education in our field. The committee may also wish to comment on the recent past history."

This is the report of the Committee. It comprises a main body with our assumptions, findings of fact, conclusions, and recommendations. There are two appendices, prepared by others, containing a great deal of specific factual information and some additional analysis.

Assumptions

MPS is a professional, scientific society whose members engage in research and teaching of the theory, implementation and practical use of optimization methods.

It is within the purview of MPS to promote its activities (via publications, symposia, prizes, newsletter), to set standards by which that research can be measured (such as criteria for publication and prizes, guidelines for computational testing, etc.), and to take positions on issues which directly affect our profession.

It is not within the purview of MPS to market software products, and MPS should not become involved in issues related to the commercial aspects of our profession except where it directly affects research and education.

The Committee is unable to make expert legal analyses or to provide legal counsel. The main body of this report is therefore written from the perspective of practitioners of mathematical programming rather than from that of attorneys skilled in the law.

MPS is an international society. However, the Committee has interpreted its charge as applying specifically to U.S. patent law and its application to algorithms. All comments and conclusions of this report should be read with this fact in mind.

Facts about patents and copyrights

The three principal forms of legal protection for intellectual property are the copyright, the patent, and the trade secret. Copyrights and patents are governed by federal law, trade secrets by state law. Setting aside the issue of trade secrets, some of the distinctions between copyrights and patents can be summarized as follows.

Type of property protected: Patents protect ideas, principally "nonobvious" inventions and designs. It is well estabished that "processes" are patentable. The Patent Office currently grants patents on algo-

rithms and software, on the basis of the ambiguous 1981 U.S. Supreme Court decision in *Diamond v. Diehr*.

Copyrights do not protect ideas. Instead, they protect the *expression* of ideas, in "original works of authorship in any tangible medium of expression." The principle that software is copyrightable appears to have been well established by the 1983 decision of the U.S. Court of Appeals in *Apple v. Franklin*.

How protection is obtained: Federal law is now in essential conformity with the Bern Copyright Convention. As a consequence, international copyrights are created virtually automatically for most works of authorship. Government registration of copyrights is simple and inexpensive to obtain.

By contrast, patents are issued by the U.S. Patent Office only after an examination procedure that is both lengthy (three years or more) and costly ($10,000 and up in fees and legal expenses). An inventor must avoid public disclosure of his invention, at least until patent application is made, else the invention will be deemed to be in the public domain. Patent application proceedings are confidential, so that trade secret protection can be obtained if a patent is not granted.

Length of protection: U.S. patents are for 17 years. Copyrights are for the lifetime of the individual plus 50 years or, in the case of corporations, 75-100 years.

Facts about algorithms

Algorithms are typically designed and developed in a highly decentralized manner by single individuals or small groups working together. This requires no special equipment, few resources, and little cost. The number of people involved is also quite large compared to the needs of the marketplace. Independent rediscovery is a commonly occurring phenomenon.

There is a long and distinguished history of public disclosure by developers of mathematical algorithms via the usual and widely-accepted channels of publication in scientific journals and talks at professional meetings. These disclosures include the theoretical underpinnings of the method, implementation details, computational results, and case studies of results on applied problems. Indeed, algorithm development is based on the tradition of building upon previous work by generalizing and improving solution principles from one situation to another.

The commercial end product of an algorithm (if there is any) is generally a software package, where the algorithm is again generally implemented by a very small number of individuals. Of course, a larger group of people may be involved in building the package around the optimization software to handle the user interface, data processing, etc. Also, others may be involved to handle functions like marketing, distribution, and maintenance.

Competition in the marketplace has been traditionally based on the performance of particular implementations and features provided by particular software products. The product is often treated like a "black box" with the specific algorithm used playing a rather minor role.

The cost of producing, manufacturing, distributing and advertising optimization software is often quite small. Even when this is not the case, it is generally the implementation of algorithms that is costly, rather than their development. Software manufacturers have a need to protect their investment in implementation, but have little need to protect an investment in algorithmic development. In the absence of patents, algorithms—like all of mathematics and basic science— are freely available for all to use.

Traditionally, developers of optimization software have protected their investments by keeping the details of their implementation secret while allowing the general principles to become public. Software copyrights are also an appropriate form of protection, and are now widely used. Moreover, despite unresolved legal questions concerning the "look and feel" of software, the legal issues of copyright protection seem to be relatively well settled.

Often an optimization package is a small (but important) part of an overall planning process. That process is often quite complex; it may require many resources and great cost to complete, and the potential benefits may be uncertain and distributed over a long time period. In such situations it is usually quite difficult to quantify the net financial impact made by the embedded optimization package.

Public policy issues

Will algorithm patents promote invention? Article I, Section 8 of the U.S. Constitution empowers Congress "To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries." Inasmuch as patents are intended to provide an incentive for invention, it seems appropriate to inquire whether patenting of algorithms will, in fact, create an incentive for the invention of algorithms.

Given the existing intensity of research and the rapid pace of algorithmic invention, it seems hard to argue that additional incentives are needed. In fact, there is good reason to believe that algorithm patents will inhibit research, in that free exchange of ideas will be curtailed, new developments will be held secret, and researchers will be subjected to undesired legal constraints.

Will algorithm patents provide needed protection for software manufacturers? Copyright and trade secret protection appear to provide the sort of protection most needed by software manufacturers. By their nature, patents seem to offer a greater potential for legal confrontation than copyrights. Instead of providing protection, algorithm patents actually pose a threat to smaller software houses lacking the resources to defend themselves in costly patent litigation. It can be argued that patents encourage an oligarchical industrial structure and discourage competition.

Is the Patent Office able to deal with algorithm patents? There is abundant evidence that the Patent Office is not up to the job. Many algorithmic "inventions" have

been granted undeserved patents, greatly increasing the potential for legal entanglement and litigation. Moreover, it seems unlikely that there will be any substantial improvement in the quality of patent examinations.

Conclusions

It seems clear from the previous discussion that the nature of work on algorithms is quite different from that in other fields where the principles of patents apply more readily. This in itself is a strong argument against patenting algorithms.

In addition, we believe that the patenting of algorithms would have an extremely damaging effect on our research and on our teaching, particularly at the graduate level, far outweighing any imaginable commercial benefit. Here is a partial list of reasons for this view:

- Patents provide a protection which is not warranted given the nature of our work.
- Patents are filed secretly and would likely slow down the flow of information and the development of results in the field.
- Patents necessarily impose a long-term monopoly over inventions. This would likely restrict rather than enhance the availability of algorithms and software for optimization.
- Patents introduce tremendous uncertainty and add a large cost and risk factor to our work. This is unwarranted since our work does not generate large amounts of capital.
- Patents would not provide any additional source of public information about algorithms.
- Patents would largely be concentrated within large institutions as universities and industrial labs would likely become the owners of patents on algorithms produced by their researchers.
- Once granted, even a patent with obviously invalid claims would be difficult to overturn by persons in our profession due to high legal costs.
- If patents on algorithms were to become commonplace, it is likely that nearly all algorithms, new or old, would be patented to provide a defense against future lawsuits and as a potential revenue stream for future royalties. Such a situation would have a very negative effect on our profession.

Recommendations

The practice of patenting algorithms is harmful to the progress of research and teaching in optimization, and therefore harmful to the vital interests of MPS. MPS should therefore take such actions as it can to help stop this practice, or to limit it if it cannot be stopped.

In particular:

- The MPS Council should adopt a resolution opposing the patenting of algorithms on the grounds that it harms research and teaching.
- MPS should urge its sister societies (*e.g.*, SIAM, ACM, IEEE Computer Society, AMS) to take a similar forthright position against algorithm patents.
- MPS should publish information in one or more of its publications as to why patenting of algorithms is undesirable.
- The Chairman of MPS should write in his official capacity to urge members of Congress to pass a law declaring algorithms non-patentable (and, if possible, nullifying the effects of patents already granted on algorithms).
- MPS should support the efforts of other organizations to intervene in opposition to the patenting of algorithms (for example, as friends of the court or with Congress). It should do so by means such as providing factual information on mathematical programming issues and/or history, and commenting on the impact of the patent issue to our research and teaching in mathematical programming. MPS should urge its members to do likewise.

26 September 1990

Appendix A

The case against "software patents"

by Brian Kahin

Brian Kahin is an attorney and consultant specializing in intellectual property, information technology, and policy development. He is currently an adjunct research fellow and director of the Information Infrastructure Project in the Program on Science, Technology and Public Policy at Harvard's John F. Kennedy School of Government. He recently prepared reports on issues surrounding the development of the National Research and Education Network for the U.S. Congress Office of Technology Assessment.

Mr. Kahin was a principal in the founding of the Interactive Video Industry Association and has served as general counsel for the Association since its inception in 1987. He is also counsel for the International Interactive Communications Society, the society for professionals in interactive media, and has recently been appointed counsel to the Federation of American Research Networks.

Mr. Kahin was instrumental in the conception and development of the EDUCOM Software Initiative — which has involved hundreds of universities and publishers in addressing issues in the creation, distribution, licensing, management, and commercialization of software in higher education — and is the original author of the widely published "EDUCOM Code" on intellectual property rights. He is currently directing the EDUCOM project on software patents.

During the past three years, Mr. Kahin has chaired subcommittees on user rights and derivative works within the Databases Committee (PTC-702) of the American Bar Association. He was formerly affiliated with the Research Program on Communications Policy at MIT, where he served in various capacities for the Research Program, the MIT Communications Forum, and Project Athena. Mr. Kahin received his B.A. from Harvard College in 1969 and his J.D. from Harvard Law School in 1976.

Mr. Kahin's telephone number is (617) 864-6606 and his electronic mail address is kahin@hulaw1.harvard.edu.

New round in an old debate

Shortly after Diamond v. Diehr [1] was decided by the Supreme Court in 1981, the twenty-year-old debate over the patentability of computer programs subsided. In that case, a 5-4 majority of the Court, by accepting the patentability of a process for curing rubber in which a computer program was the major component, had found their way around earlier misgivings about the patentability of mathematical algorithms. [2]

The Patent Office, which had long resisted granting software patents and struggled regularly with the Court of Customs and Patent Appeals (CCPA) over the patentability of program-related inventions, subsequently underwent a change of heart. After pending appeals to the CCPA were decided in 1982, not a single case concerning a rejected application for program-related patent was heard by either the CCPA or its successor, the Court of Appeals for the Federal Circuit (CAFC), for the following seven years. As the debate subsided, the number of software patents granted by the Patent Office began to grow (Table 1). By early 1989, the trickle had become a torrent.

Table 1. "Software Patents" Granted by USPTO: 1981-1989

1981	**21**
1982	**52**
1983	**64**
1984	**136**
1985	**153**
1986	**187**
1987	**227**
1988	**183**
1989	**(579)**

(extrapolation from
1st four months of 1989: = 3 X 197)

Source: Electronic Data Systems, "Software Patent Indexes — January 1970 thru April 1989," prepared for the Computer Law Committee of the State Bar of Texas. [3]

As these new software patents have drawn attention, the debate has flared up. It now takes place in an environment transformed by the microcomputer, in which a fast-growing, mass-market publishing industry has emerged. Software has found its way into almost every office and into tens of millions of homes, performing commonplace and extraordinary functions through a seemingly infinite repertoire of processes.

In this new round of debate, the hardware and software industries have switched sides. In 1978, when Parker v. Flook was before the Supreme Court, ADAPSO (representing software interests) filed an amicus brief favoring patentability, while CBEMA (representing hardware interests) filed a brief opposing patentability. Since the early days of computers, the hardware industry (including IBM) had opposed patentability of software in the belief that software patents would inhibit software development and so limit the market for hardware. Software developers, while endorsing patentability in theory, seemed uninterested in pursuing patents in practice. [4]

In the meantime, hardware manufacturers, recognizing the growing economic importance of software, have come to see software as an extension of their hardware business — rather than merely as a complementary product produced by third parties. Major hardware firms, accustomed to patent-oriented strategies and armed with in-house patent counsel, have filed for and received hundreds of software patents. By one count, IBM leads with 264 over a nine-year period, followed by Hitachi with 69. [5]

The new software publishing industry, which developed without using patent protection, has suddenly found itself vulnerable to charges of patent infringement. Humble origins, rapid growth, strong market orientation, international acceptance of copyright, and the costs and uncertainty of patenting — all led publishers to rely on copyright. [6] Even today, only a handful of patents have been issued to established software publishers. But having seen evidence of widespread patenting of software processes, software publishers are fearful that patents will increase development costs, inhibit creativity and innovation, and embroil the industry in litigation. [7]

While the Software Publishers Association has not taken a formal position on the issue, Executive Director Ken Wasch has spoken strongly against software patents. [8] At this point, few software developers understand the full impact of the patent system on traditional practices, and much of their frustration and anger is vented at the seeming ineptitude of the Patent Office. Indeed, both sides of the debate share in widespread concern that the Patent and Trademark Office lacks the expertise and resources to process software patents properly.

While the earlier debates focussed on the patentability of mathematical algorithms — *i.e.*, code-level processes, much of the current debate concerns the user interface. For example, computer-implemented analogs of conventional practices such as footnoting [9] and redlining [10] are receiving patents which in many cases appear broad enough to block most if not all computer implementations of these practices. The late appearance of patents at this level is a consequence of the user-oriented design and functionality of microcomputer software.

Meanwhile, the academic literature on software patents has waned. [11] There is none that takes into account the scope and breadth of the patents that have been issued since Diamond v. Diehr. In the last three years, considerable practitioner-oriented literature has appeared (especially in The Computer Lawyer, [12]) and called attention to the wide range of software processes now susceptible to patenting. Largely unmindful of policy issues, this literature presents patent as a powerful tool for protecting investments in software development. It is oblivious to the criticisms and sense of crisis that has arisen within the software publishing industry since last year — for which one must look to the trade and general press. [13]

The patent bar stoutly denies that there is such a crisis and argues that the apparent problems are merely typical of patenting in any new technology until the Patent Office acquires sufficient expertise in the subject matter. [14] Of course, the patent bar has a vested interest in expanding the scope of patentable subject matter. Patent practice is the only speciality within the law requiring special certification and training (including a degree in a science or engineering discipline), and patent practitioners constitute a close-knit community which gravitates around a federal agency in Crystal City, Virginia.

The inauguration of the Court of Appeals for the Federal Circuit (CAFC) in 1983 has enhanced the special status and insularity of patent practice. In the interest of promoting uniformity in the law, the CAFC is assigned jurisdiction over all appeals from patent decisions in the District Courts, as well as appeals from Patent Office decisions (which were formerly heard by the CCPA). With panels in patent cases led by members of the patent bar, the CAFC has established a strong pro-patent record and has greatly

strengthened the presumption of patent validity. [15] Even a sympathetic observer has noted the "evangelical fervor" with which the CAFC has pursued its course of action. [16] The CAFC, and the Reagan Administration's general support for patents, created an encouraging atmosphere for the Patent Office's foray into software.

Capability of the Patent Office

In theory, given enough resources, it should be possible for the PTO to process software patents expeditiously within the present framework of the patent system. However, the 1966 Report of the President's Commission on the Patent System observed:

The Patent Office now cannot examine applications for programs because of the lack of a classification technique and the requisite search files. Even if these were available, reliable searches would not be feasible or economic because of the tremendous volume of prior art being generated. Without this search, the patenting of programs would be tantamount to mere registration and the presumption of validity would be all but nonexistent.

To most of the software industry, these observations hold true today. Despite the PTO's embrace of software patents after Diamond v. Diehr in the early 1980s, it has made no apparent effort to address these problems in any systematic manner. The classification scheme remains undeveloped, [18] and the problem of locating prior art has only been exacerbated by the long delay. Ironically, the PTO now maintains that there is no problem, [19] and the presumption of patent validity has been greatly strengthened by decisions of the CAFC.

It is difficult for the PTO, like other government agencies, to find resources to meet new challenges, and the 1966 President's Commission in part echoed the PTO's own anxiety about launching into an uncharted, seemingly fathomless area. In theory, if only sufficient funding were available, the Patent Office could hire the right people, build a comprehensive library to reference prior art, develop a sophisticated classification system, and invest in the best automation. But while the funding of the Patent Office has more than doubled in the past seven years and average processing time has been reduced to 18-19 months, average processing time for software patents remains substantially higher — around 30 months. [20] Indeed, since it has recently been taking the Patent Office around two months after the grant to actually publish patents, the effective processing period is, in certain respects, 32 months. [21] Despite the longer processing period, there is widespread concern about the quality of the review process because so many of the patents issued look obvious to industry observers. Indeed, the lack of reported appeals from Patent Office decisions since 1982 suggests that most applications for software patents are eventually granted.

One frequently cited problem is that the Patent Office does not accept a computer science degree as a qualifying degree for attorney registration. Even if it did, there is a question of how well the Patent Office can attract and hold expertise in an area where industry demand is high. Difficulty in attracting qualified personnel naturally creates problems in applying the nonobviousness standard, since less qualified examiners will tend toward a lower standard in determining the hypothetical "person having ordinary skill in the art." Less qualified personnel are also more likely to be influenced by sophisticated patent attorneys in the ex parte review process, and the point system for promotion of personnel rewards dispositions, which encourages the granting of patents in close cases.

The problem of locating prior art noted by the President's Commission is first of all due to the lack of a base in the patent records themselves. In a mature technology, the patent records with their classification system and cross-referencing provide the most useful base of prior art. However, since history passed the Patent Office by thirty years ago, only an infinitesimal portion of the art of computer programming is to be found in the patent database. In most fields, the prior art in the patent base is supplemented by published technical literature, but in computer programming the literature is scant and unorganized. There was long a dearth of scholarly literature in the field, and there is still no bibliographic database.

Nor does the Patent Office maintain a library or archive of software that might provide a record of the prior art. However, software alone does not qualify as a "printed publication" establishing prior art, and further evidence would be required to show that the software was used publicly. Locating and documenting such evidence is often expensive and is usually undertaken only in the course of patent litigation.

In addition, many programs used commercially are licensed subject to trade secret restrictions against reverse engineering. When a software process is hidden and contractual restrictions effectively preclude its disclosure, it may be considered suppressed and therefore not qualify as prior art. [22] On the other hand, it may qualify as prior art if the license is not effective in precluding reverse engineering — as may well be the case with common shrink-wrap licenses. [23]

Nature of the software industry

Since the patent system operates under the principle of winner-take-all, only the very first to invent gets the patent. The efforts of the losers are wasted in that they must design around the patent (if that is possible) or pay the winner for the privilege of using the patent.

If one knew who had invented what when it happened, this would not be so bad. However, patent applications remain secret unless and until the patent issues. This secrecy permits the applicant to maintain trade secret protection if the patent application is not granted. However, with no way of knowing what patents are in the pipeline, a software developer can never be assured of avoiding patent infringements. While this

problem is endemic to the patent system, factors combine to make it exceptionally severe in the case of software.

Product development cycles in the software business are typically much less than the current 32-month average time between application and the publication of patent grant. Indeed, the product life cycle may well be less than 32 months. Even 20 months, the current average for all patents is a very long time in the software business. [24]

At the same time, the potential for waste is uniquely severe in the software industry because of the sheer number of players. The highly decentralized nature of the industry makes it likely that many different individuals or firms will come up with similar new processes at about the same time. Structurally, the software industry resembles "industries" traditionally protected by copyright, in which there are a large number of diverse participants who accept independent creation as a natural defense and risk.

Thus the blindsiding problem is, in the first instance, a product of the number of players times the length of the pendency period. The third factor, the complexity of software — the growing number of patentable processes in software products — will be discussed below.

The highly decentralized structure of the software industry may also be an accident of history — of the virtual absence of patenting until recent years. Investment and development practices were designed simply to avoid unauthorized copying — the essence of trade secret and copyright. Trade secret and copyright were widely accepted, although there is certainly controversy about copyright protection at higher levels of abstraction — the area of "look and feel" and "structure, sequence, and organization." But with widespread patenting, the rules seem to have changed in the middle of the game, albeit as the effect of court decisions and patent office policy made seven years ago.

Part of the delayed reaction is due to a belated understanding of the significance of patents. Patents are usually portrayed by lawyers simply as another form of protection which developers are free to take or leave depending on costs, the likely value of the patent, and other strategic considerations. [25] But that is only half the picture. The costs of doing business under the patent environment include the costs and risks of avoiding infringement, which are borne by every developer regardless of whether the developer chooses to apply for patents. All must adapt to a new paradigm of operation in which defensive research precedes creation.

If software had clearly been patentable from the outset, there would be no surprise, no defeated expectations. But now it appears that the industry may have to be reshaped to fit the patent system and that the rapid development of software products may have to be slowed to fit the review, processing, and publication cycle of the Patent Office. Most important, the low barriers to entry which have characterized the software industry will be raised significantly by the costs of operating under the patent system.

Much of software development has been a cottage industry, in which independent genius flourished unencumbered. Visicalc, 1-2-3, WordPerfect, and many other microcomputer programs were initially created by individuals and small teams. Software development requires no materials, no special facilities, and no special tools. To design it is to build it.

While it cannot be proven empirically, it is widely accepted within the software community that there are diseconomies of scale in software development. [26] Under a patent regime, software development is no longer a place for individual authors with good ideas and the skills to transform them into marketable software. It is instead an industrial enterprise, in which the cost and risks of avoiding and claiming patents are part of the cost of doing business.

True, it will remain possible for individuals and small companies to pursue strategic patents themselves, which they may in turn license to large companies. But they will increasingly need to license from other patent holders in order to assemble a marketable product. Only companies with a full complement of patents for cross-licensing and ample financial resources to absorb the continuing costs and risks of pursuing and defending against patents will be able to innovate rationally. Investments at the venture capital level will necessarily be diverted from developing solid market-oriented products to speculation in strategic patents.

Although cross-licensing allows efficient, competitive exploitation of patents in industries where there are relatively few firms of roughly similar size, cross-licensing will not work for the many thousands of small firms and tens of thousands of individuals in the software industry — because these small players have little or nothing to bring to the table. The vision of cross-licensing as a solution to the problem of software patents implicitly assumes a wholesale shakedown and restructuring of the industry. In fact, since software publishers hold so few patents, it may imply the death of software publishing as an independent industry and its absorption by hardware manufacturers who have the arsenals needed for meaningful "protection" in the patent world.

Accordingly, "defensive patenting," the strategy of acquiring patents for cross-licensing is an option only for the largest firms, those that can afford to acquire a meaningful portfolio. While defensive patenting may of value in whatever cross-licensing scenario emerges over the long run (or in selling out the company to a hardware manufacturer), it is of limited value in the short run. First, because of the time required to prepare and pursue a patent application. Secondly, because the particular "defensive patents" must be needed by the company that holds a needed strategic patent. Defensive patents are completely ineffective against firms such as Refac International which have no interest in producing software themselves.

Since a patent, unlike copyright, includes the exclusive right to control the use of the product or process, a patent holder may pursue the users of an infringing product or process independent of the manufacturer or seller. In this respect, too, software publishers may be victims of their own success. Not only is their product likely to be used by large companies with deep pockets, but volume licenses are common, providing patentees with an easy trail to infringing users. This gives the patentee enormous leverage over the publisher. Users will be reluctant to deal with publishers that expose them to lawsuits and that are unable to indemnify them for patent infringements. Once they understand this problem, users, especially large corporate users, will be reluctant to acquire software from small companies.

Small developers currently have the option of distributing through large publishers for which they receive a royalty of 10-20% of sales. This option, too, will be limited and foreclosed. First, the publisher's customary royalty is likely to be reduced to reflect the publisher's risk of patent infringement. But once royalties must be paid to patent holders, the developer's margin disappears very quickly. A "reasonable royalty" is commonly used as a measure of damages for patent infringement, especially when the patent holder is not working the patent. Although awards of "reasonable royalties" vary, the trend is upward with recent CAFC decisions affirming rates of 5 to 33 ⅓%. [27] At these rates, royalties on only one or two patents put the developer out of business.

Altogether, software developers face new costs of operation at three different categories:

At the first level are the costs of searching and analyzing prior art to avoid patent infringement. A precautionary search and report by outside patent counsel may be expected to cost $2,000. Bear in mind that this cost and other calculations are per process rather than per product. Other costs include the risk of infringing patents which cannot be found because they are still in the pipeline or because of the deficiencies of the classification system (or the PTO's misclassification of particular patents). [28]

The second set of costs are the license fees that must be paid to holders of valid patents, including the costs of determining the necessity and value of the license, negotiating the license, and reporting to the patent holder. If the patent holder refuses to license, there are costs of designing around the patent, if that is possible. Otherwise, there will be the greater costs of reconceiving or even abandoning the product. These constitute the direct costs imposed by the patent monopoly which are theoretically justified by the incentive to the patentee and the value of the knowledge contributed to the public domain.

The third set of costs are those incurred only when the developer chooses to file for patents. These include the usual first-level costs (searching for prior art) plus all the additional expenses of preparing, filing, negotiating, and maintaining a patent. In addition to the $10,000 to $15,000 that may be expected for a patent attorney's services, considerable time will be required from the developer's staff to adequately prepare and prosecute the application. Administrative fees over the life of the patent will exceed $3,000. Thus, the real cost can easily come to $25,000 or more. And this will run substantially higher if foreign patents are also sought.

The notoriously high costs of patent litigation must be borne by both sides — and so must be attributed to both the first and third categories. Just to get through the discovery phase of a lawsuit is likely to cost each side $150,000, and a full trial will cost each $250,000 on up into millions — in direct costs, not including the staff time absorbed in litigation. While a small patent holder may be able to secure a law firm on a contingency basis or sell an interest in the patent to speculators, the defendant has no such options.

Litigation also involves the possibility, and further expense, of an appeal. All appealed patent cases now go directly to the Court of Appeals for the Federal Circuit, where the panels include patent lawyers turned judges. Whereas patents once fared poorly in the circuit courts, the CAFC has found patents to be both valid and infringed in over 60% of the cases that have come before it.

These high costs give patentees considerable leverage over small firms, who will, as a practical necessity, pay a license fee rather than contest a dubious patent. The patent holder can then move on to confront other small firms, pointing to such license agreements as acknowledgments of the patent's validity and power. This tactic has a snowballing effect that can give the patent holder the momentum and resources to take on larger companies when the time is ripe.

All these costs must be paid by someone. In the short run, they may come out of the software industry's operating margins, but in the long run, as the industry shakes out, they will be borne by users. At the same time, the shakeout will mean a loss of diversity — in innovative firms and in the products available to consumers.

Nature of software

The costs and risks associated with patenting have grown geometrically as software has become more sophisticated and complex. But, again, the impact was not felt until recently. In the meantime, driven by a highly competitive software market, software developers added function after function heedless of whether somebody else might have implemented it first.

Applications software that consisted of a couple thousand lines of code when originally released in the early 1980s may now contain ten thousand lines. Dan Bricklin, author of the original spreadsheet, Visicalc, estimates that a modern applications program may contain thousands of processes which are patentable under present Patent Office standards. [29] Although most of these processes are now part of the public domain, at this scale the odds of infringement are probably high.

The highly integrated nature of software makes the prospect of infringement especially disturbing. It is usually difficult to excise an offending process because each process will be interconnected to the remainder of the program package at so many points. Several iterations of redesign and testing may be required in the removal of an infringing process.

Software is often built on top of other software, so that a patent problem at one layer may preclude use of the high layers. In this way, applications software rides on operating system software, and many programs are sold with run-time modules of programs which operate at a lower level of functionality. A patent infringement in an authoring system, for example, may jeopardize all programs created with that system.

The evolution of complex, layered software also raises problems in applying the standard of nonobviousness. Where is the person of ordinary skill in the art to be found? In the elite few who design microcomputer software or in the ranks of 700,000 computer programmers? In theory, the Patent Office should acknowledge specialities within the development process that differentiate code writers from higher-level designers. But, as noted, the Patent Office does not even recognize computer science as a qualifying degree, so it seems unlikely that it will recognize finer degrees of specialization.

A related problem is the breadth of the claims allowed by the Patent Office for software processes at the user-interface level, often at a level of abstraction that is shocking to the uninitiated. [30] Even though the disclosure is in terms of a flow chart rather than code (which enables the applicant to maintain trade secret protection at the code level), there is often great disparity between the particular implementation disclosed and the broadest claims in the patent. All that need be disclosed is the "best mode contemplated by the inventor of carrying out his invention," [31] which may be merely one of many.

Some of these patents appear to preempt automation of common functions such as footnoting and redlining. [32] Others appear to preempt obvious procedures for entering data, [33] providing on-line help, [34] or other fundamental user interface elements. For years software developers have been routinely implementing software analogs or simulations of common office functions, writing and bookkeeping procedures, learning strategies, and other human activities. Now the Patent Office is proceeding as if the first to computerize any such function merits a 17-year monopoly over any software implementations of the function.

In part, this is an issue of what is obvious to whom. But it also reflects the sweeping aside of judicial doctrines barring patentability for "methods of doing business" and "mental steps." [35] For example, a patent held by Merrill Lynch on its Cash Management Account system was upheld — precisely because it was implemented on a computer rather than by hand. [36] In effect, this 1983 decision stood Diamond v. Diehr on its head, since Diamond v. Diehr established that a computer program did not render an otherwise patentable process unpatentable. These decisions have been accepted by the Patent Office [37] — although the Supreme Court has yet to rule on these matters.

As is frequently noted, software processes can be implemented in hardware as well. But the problem is not software versus hardware — but abstract processes which are now typically implemented in software. This problem is not even limited to computers. For example, a patent issued in 1979 for an "interactive teaching machine" claims not only a particular machine but also the process of showing introductory video segments, presenting the viewer with a choice of decisions, and then showing the viewer the likely outcome of the chosen decision. [38] Although the patent discloses a clumsy-looking combined videotape deck and television, the broad process claim covers a basic mode of user interaction common in interactive videodisc design.

This example also illustrates the great difficulty of locating many "software patents."

When such broad claims are allowed, the traditional quid pro quo — the grant of a limited patent monopoly for revealing what otherwise might be kept as a trade secret — is lacking. The process claimed is self-evident; it cannot possibly be concealed. Furthermore, what is claimed often appears to be functionality rather than a process — the two can be difficult to distinguish in the user interface area. [39] Although the particular implementation may not be self-evident, implementation may be achieved in many different ways with know-how and hard work. Thus, the particular implementation disclosed in the patent does not necessarily add anything of significance to public knowledge.

Pragmatism and policy

The traditional quid pro quo has plainly failed in practice. No one in the industry looks to patents to learn state-of-the-art programming and design. Even now, what comes through the patent system is almost by definition not state-of-the-art because it is already nearly three years old. (Had software been patented from the start, this might be different: There might now be only one word processor, one spreadsheet, one database manager, all with less functionality than those presently on the market — but all state-of-the-art by definition.)

Nor is there evidence that the fundamental quid pro quo — the incentives provided by the seventeen-year patent monopoly — applies. As the President's Commission on the Patent System reported in 1966:

It is noted that the creation of programs has undergone substantial and satisfactory growth in the absence of patent protection and that copyright protection for programs is presently available.

After 23 years of dramatic growth and worldwide acceptance of copyright, these words hold all the more true today.

Even if the nature of the software development could not be differentiated from other industrial development, this situation is historically unique. The patent system has taken effect in many industries which previously had no intellectual property protection (other than trade secret), but there is no other case where the patent system has been imposed on an industry already protected by copyright. The industry is healthy, competitive, and prolific. It is hard to imagine that the full press of a patent regime could propel it faster. Obviously, the software industry is not broke — but, nonetheless, it is slowly being "fixed."

Indeed, from a patent perspective, there appears to be overinvestment and disorderly development in the industry. This perspective sees wasted resources under a copyright-only regime because of the uninhibited duplication of effort.

The patent system's answer to this chaos is a technocratic vision that would rationalize progress in the industry by reducing the number of players and creating a massive database of patents that represents the accumulated knowledge of the industry at any one time. The pace of development would be slowed to fit the cycles of the patent system and to reduce waste. In the long run, a few healthy giants would compete on a global scale, bidding for the ideas and loyalty of inventive individuals, and developing the technology with the knowledge and wisdom that comes from vertical integration and aggregation of resources.

The problems with this vision are first the problems inherent in the patent systems — those that have been discussed and the fact that patent holders may act in their own self-interest to impede progress. Experience in the early years of the auto industry, the aircraft industry, and the radio industry all demonstrate the potential of patents for blocking progress. These problems are accepted as the price of providing incentive, but here no incentive is needed. Moreover, these problems are exacerbated by special characteristics of software, software development, and the present software industry — whether these are inherent characteristics or a consequence of the long absence of patenting.

But there is a deeper problem in the technocratic vision and the likely consolidation of the industry, which was not evident when computers were known only for "data processing." As the computer becomes ubiquitous and software grows more versatile and sophisticated, the "universal machine" becomes ever more universal, ever more adept in its mediation of human experience. Increasingly, software is designed to render the technology transparent, to satisfy the specific human needs and goals — which are not concerned with using the computer, but with the shaping, expression, and delivery of information. New "content- driven" software shows how the computer program is becoming a multimedia vehicle for human expression and communication.

The collision between copyright and patent principles at the user interface level has received considerable attention in recent years — although usually in defining the limits of copyright in "look and feel" and "structure, sequence, and organization." But whatever the fit of the particular software feature into copyright or patent, what is increasingly at stake is the generation and flow of information to and from human beings. At the atomic level, information may not be patentable (it may not even be protectable under copyright), but it rides on processes that are now susceptible to patenting — just as higher level software processes ride on lower level processes.

Information per se is traditionally the substance and territory of copyright — which, in the interests of maintaining free commerce in ideas and maximizing freedom of speech, forbids only direct appropriation of particular expression. Patent, by contrast, is an all-pervading property right, which is measured by the full scope of the claimed idea rather than the inventor's particular implementation (the expression). [40]

The intelligent ordering of information is at the very heart of grammar, rhetoric, and graphic design. Should information which is interactive rather than linear be subject to the pervasive restraints of patent? Should human expression which is assembled or assimilated with the aid of a computer be restrained by patents? If the computer is seen as extension of the human mind rather than vice versa, it is difficult to answer yes.

Notes:

[1] 450 U.S. 175

[2] Gottschalk v. Benson, 409 U.S. 63 (1972); Parker v. Flook, 437 U.S. 584 (1978).

[3] These figures probably vastly underestimate the number because of the limitations of the EDS methodology and PTO's classification system. For example, the footnoting patents cited in note [9] below are not included.

The other known study, by John T. Soma and B. F. Smith ("Software Trends: Who's Getting How Many of What? 1978 to 1987," Journal of the Patent and Trademark Office Society, Vol. 71, No. 5, May 1989, pp. 419-432), is even more limited. It focuses solely on patents originally classified or cross-referenced to subclass 300 ("Programming Methods or Procedures") of Class 364. It therefore shows only 262 "software patents" issued during the ten-year period from the beginning of 1978 through the end of 1987.

[4] See Michael C. Gemignani, "Legal Protection for Computer Software: The View from '79," Rutgers Journal of Computers, Technology, and the Law, Vol. 7, p. 269-312, 307ff.

[5] Electronic Data Systems, note [3] above, figures for 1980 through April, 1989.

[6] Reliance on copyright protection does not exclude the possibility of patenting aspects of software which cannot be protected by copyright. However, the patenting process appeared costly and uncertain. Software publishers also attempted to use trade secret protection, but in a mass market they had difficulty establishing the contractual relationship required by trade secret. See David A. Rice, "Trade Secret Clauses in Shrink-Wrap Licenses," The Computer Lawyer, Vol. 2, No. 2, February 1985, pp. 17-21.

[7] William M. Bulkeley, "Will Software Patents Cramp Creativity?" Wall Street Journal, March 14, 1989, p. B1.

[8] Lawrence M. Fisher, "Software Industry Is in an Uproar Over a Rush of Patents," New York Times, May 12, 1989, p. A1.

[9] Patent Nos. 4,648,067 and 4,648,071, issued March 3, 1987.

[10] Patent No. 4,807,182, issued Feb. 21, 1989.

[11] Perhaps the last article of significance is Donald Chisum's, "The Patentability of Algorithms," University of Pittsburgh Law Review, Vol. 47 No. 4, pp. 959-1022, in which Chisum attacks the Supreme Court's distinction between mathematical and computer algorithms as untenable and argues that Benson and Flook were wrongly decided.

[12] *E.g.*, John P. Sumner, "The Versatility of Software Patent Protection: From Subroutines to Look and Feel," The Computer Lawyer, Vol. 3, No. 6 (June 1986), p. 1; Travis Gordon White and Richard T. Redano, "Patent Opportunities for Software-Related Subject Matter," The Computer Lawyer, Vol. 4, No. 7 (July 1987); John R. Lastova and Gary M. Hoffman, "Patents: Underutilized Leverage for Protecting and Licensing Software," The Computer Lawyer, Vol. 6, No. 5 (May 1989).

[13] Vince Canzoneri, "Software Enters the Age of Patent Protection," Computer Update July/August 1988, p. 32; "Software Patents: 'A Horrible Mistake'," Softletter, September 1, 1989; Rachel Parker, "Refac's Unworthy Patent May Rally the Rest of the Industry," Infoworld, August 7, 1989, p. 42; Steve Gibson, "Attorneys' Glee Over Patent Squabbles Breeds Pessimism," Infoworld, October 9, 1989, p. 26. See also the articles cited in notes 7. and 8. above.

[14] *E.g.*, David Bender, letter to the editor, New York Times, June 8, 1989.

[15] The presumption of patent validity can now be overcome only by clear and convincing evidence. Lindemann Maschinenfabrik GmbH v. American Hoist & Derrrick Co., 730 F.2d 1452 (CAFC 1984).

[16] Robert L. Harmon, Patents and the Federal Circuit (Washington, DC: Bureau of National Affairs, 1988), p. vii.

[17] President's Commission on the Patent System, "To Promote the Progress of Useful Arts," 1966, p. 13.

[18] "Computer processes are not classified within USPTO's patent classification system in any readily identifiable set of classes and subclasses." Letter from Jeffrey Samuels, Acting Commissioner of Patents and Trademarks to Congressman Robert W. Kastenmeier, November 1, 1989, p. 2. The letter goes on to cite figures from the Soma and Smith study (note [3] above) but acknowledges, "We would have no way of knowing how accurately these numbers reflect the total number of patents having claims drawn to computer processes."

[19] *Ibid.*, p. 5.

[20] Calculated as the average for the 54 software patents issued in April, 1989, as identified by Electronic Data Systems, note [3] above.

[21] In the case of the Pardo patent (No, 4,398,249) for "natural order recalc," the application was filed in 1970 but not granted until 1983! The current assignee, Refac International, has filed an infringement action against six major spreadsheet publishers.

22] George H. Gates III, "Trade Secret Software: Is It Prior Art?," The Computer Lawyer, Vol. 6, No. 8 (August 1989).

[23] See Vault v. Quaid, 655 F.Supp. 750, E.D. LA. (1987), affirmed, 847 F.2d 255 (1988), in which the District Court summarily characterized the shrink-wrap license in question as an unenforceable contract of adhesion. The Court found copying for the purpose of reverse engineering to be sanctioned by Section 117 of the Copyright Act. See also, Steven W. Lundberg and John P. Sumner, "Patent Preemption of Shrink-Wrap Prohibitions on Reverse Engineering," The Computer Lawyer, Vol. 4, No. 4 (April 1987).

[24] A partial solution to the long pendency period would be to publish the patent application after 18 months, as is done in other countries.

[25] *E.g.*, "Patent Protection For Computer Software: The New Safeguard," title of seminar sponsored by Prentice Hall Law & Business, Washington, DC, September 13-14, 1989. Some would argue that even the word "protection" obscures the global effects of the patent system on all players, whether or not they choose to patent.

[26] Dan Bricklin, MIT Communications Forum: "Software Patents: A Horrible Mistake," March 23, 1989.

[27] Gary M. Hoffman, "Computer Software Patents Scope of Protection," in Michael S. Keplinger and Ronald S. Laurie, eds., *Patent Protection for Computer Software: The New Safeguard* (Englewood Cliffs, NJ: Prentice Hall Law & Business, 1989), pp. 128-9.

[28] Because of the dubious validity of software patents, developers may also wish to locate and document prior art independent of the patent record. As indicated earlier, in the absence of evidence in the form of a printed publication, this is likely to be an expensive undertaking. However, it is a way of buying some certainty.

[29] The Patent Office has declined to enter into any such estimate. See letter from Acting Commissioner Jeffrey Samuels answering Congressman Kastenmeier, November 1, 1989, p.4 (answer to Question 4).

[30] *E.g.*, Claim 1. from Patent No. 4,674,040, "Merging of Documents," issued June 16, 1987:

1. A method for merging a portion of one document into another document, said method comprising:

(a) including a reference in said another document to said portion; and

(b) causing said portion to be merged with said another document and displayed in merged form.

[31] 35 U.S.C. Section 112.

[32] See notes [9] and [10] above.

[33] No. 4,646,250, issued February 24, 1987, "Data Entry Screen."

[34] No. 4,648,062, issued March 3, 1987, "Method for Providing an On Line Help Facility for Interactive Information Handling Systems."

[35] In re Musgrave, 431 F.2d 882 (CCPA 1970) (mental steps).

[36] "The product of the claims of the '442 patent [No. 4,346,442] effectuates a highly useful business method and would be unpatentable if done by hand." Paine, Webber v. Merrill Lynch, 564 F.Supp. 1358, 1369 (Del. 1983).

[37] Gerald Goldberg, "Patent and Trademark Office Report on Patentable Subject Matter: Mathematical Algorithms and Computer Programs," U.S. Patent and Trademark Office, September 1989.

[38] Patent No. 4,170,832. Claim 11 reads:

11. A method of operating audiovisual teaching equipment including a stored motion video program having an introductory life-like scene segment leading to identified choices and plural motion video life-like scene segments showing the likely result of such choices, comprising:

projecting said introductory life-like segment and identified choices;

terminating projecting of said motion video program automatically in response to the end of said introductory segment and identified choices;

selecting one of said identified choices by manual actuation; and

automatically projecting in response to said selecting step only one of said plural motion video life-like segments displaying the likely result of said selecting step.

[39] Mere functions are traditionally not patentable. Denning Wire and Fence Co. v. American Steel & Wire Co., 169 F 793 (CA8 Iowa, 1909).

Section 112 of the 1952 Patent Act provides for claims in the form of "means plus function," and it was once considered preferable to claim software patents as an apparatus (a hardware circuit or a general purpose computer) with specific functionality rather than as a process. However, the courts and the Patent Office have tried to discount such differences in form.

[40] This generalization is subject to the qualification that under the doctrine of equivalents, courts may occasionally look beyond the literal claims to "equivalents." This may involve looking to the "invention as a whole" including the implementation. See Ronald S. Laurie and Jorge Contreras, "Application of the Doctrine of Equivalents to Software-Based Patents," in Michael S. Keplinger and Ronald S. Laurie, eds., *Patent Protection for Computer Software: The New Safeguard* (Englewood Cliffs, NJ, 1989).

Appendix B

Against Software Patents

by

The League for Programming Freedom

The address of the League for Programming Freedom is 1 Kendall Square # 143, P.O. Box 9171, Cambridge, Massachusetts 02139; its telephone number is (617) 243-4091, and its electronic mail address is league@prep.ai.mit.edu.

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.

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

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.

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 saw a point in writing 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". This technique is used when there are overlapping windows; the contents of a window that is partly hidden are saved in off-screen memory so they can be put back quickly on the screen if the obscuring window disappears (as often happens).

In fact, the technique of backing store was used in an earlier MIT project, the Lisp Machine System, before AT&T applied for the patent. But the Lisp Machine developers did not publish anything mentioning the use of backing store until the programmers' reference manual was written some years later. They expected that any window system developer would have the same idea, given that the memory of the computer was large enough to make the idea practical. (Earlier window systems, such as those at Xerox, did not use backing store because the computers in use had insufficient memory space to spare any for this purpose.)

Without a publication, the use of backing store in the Lisp Machine System may not count as prior art to defeat the patent. So the AT&T patent may be enforceable, 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 being threatened by Cadtrak as well). The X Windows Project was intended to develop a window system that all developers could use freely. Because of software patents, this public service goal seems to have been thwarted.

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 journal, *IEEE Computer*. 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 is still in the public domain, using it means risking a lawsuit. And implementing the algorithms found in 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 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*. (This information comes from a paper being written by Professor Samuelson of the Emory School of Law.)

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.

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.

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.

This special edition is devoted to the report of the Committee on Algorithms and the Law. The next regular O P T I M A will appear in mid-summer, 1991.

Books for review should be sent to the Book Review Editor, Prof. Dr. Achim Bachem, Mathematiches Institute der Universität zu Köln, Weyertal 86-90, D-5000 Köln, West Germany.

Journal contents are subject to change by the publisher.

Donald W. Hearn, EDITOR
Achim Bachem, ASSOCIATE EDITOR
PUBLISHED BY THE MATHEMATICAL PROGRAMMING SOCIETY AND PUBLICATION SERVICES OF THE COLLEGE OF ENGINEERING, UNIVERSITY OF FLORIDA.
Elsa Drake, DESIGNER

O P T I M A
MATHEMATICAL PROGRAMMING SOCIETY

303 Weil Hall
College of Engineering
University of Florida
Gainesville, Florida 32611-2083 USA

FIRST CLASS MAIL