Software Patents

A (Wary) Defense

By Martin Campbell-Kelly

"The patent system is in crisis," the science writer James Gleick recently told readers of *The New York Times*. "A series of unplanned mutations have transformed patents into a positive threat to the digital economy."

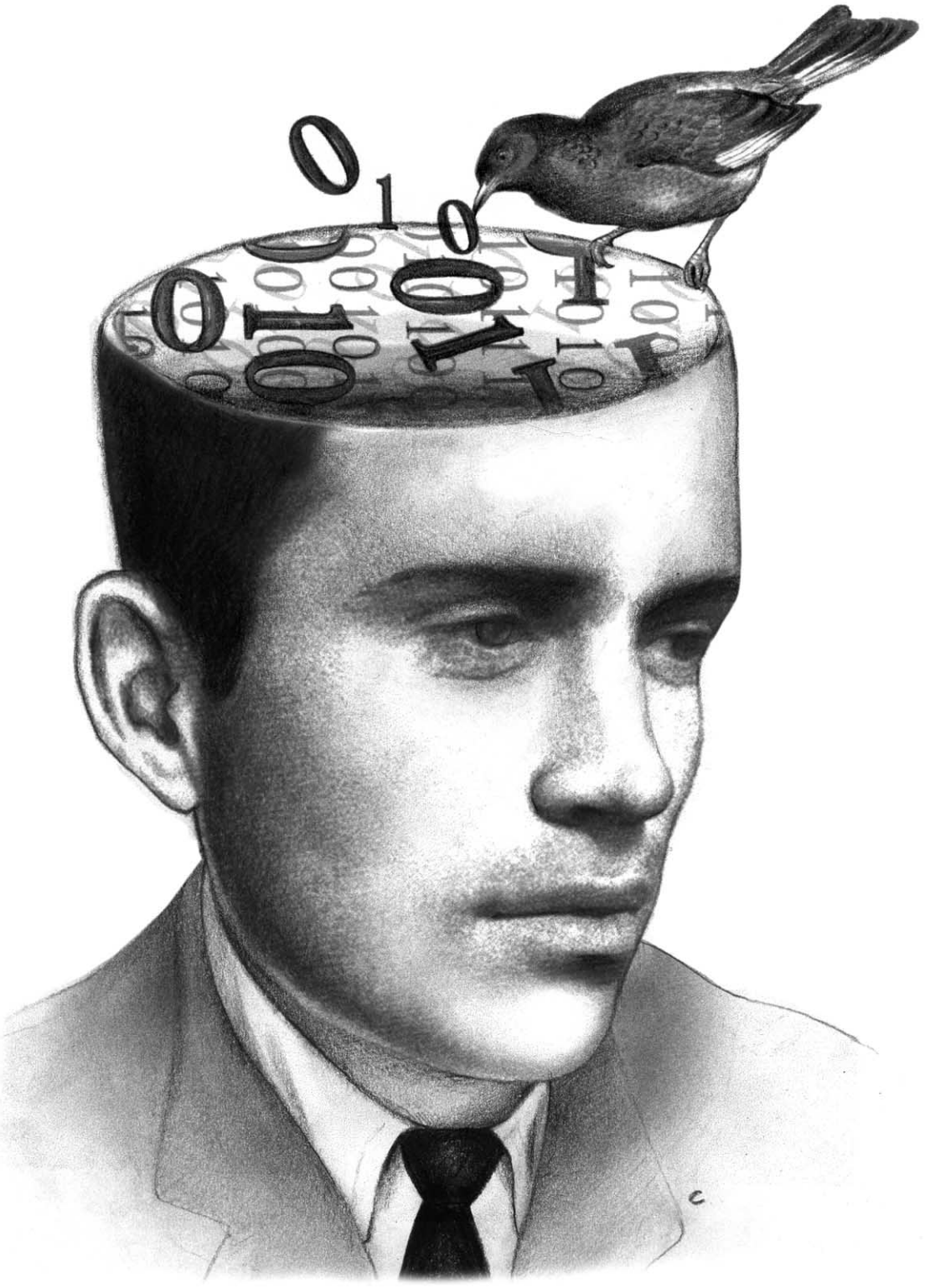
Gleick is not alone in raising the alarm. Among others, Lawrence Lessig, an influential Stanford University commentator on public policy toward digital technology, talks darkly of patent "thickets" that have become a barrier to entry for both small commercial software developers and the "open source" community – volunteer programmers who develop software to share with others at no charge. Software makers, the argument goes, are in danger of becoming caught in headlights of the patent juggernaut – frozen in fear of inadvertently infringing on others' property rights.

Patent protection is, in general, problematic because it limits the diffusion of inventions and creates legal uncertainty for those pursuing related innovation. It is especially problematic for software, critics say – in part because the embodied intellectual property can be protected with copyrights alone, in part because the software industry is simply too central to economic growth to risk inhibition in defense of property rights.

There is one point on which almost everyone who has followed the controversy agrees: the system for protecting the intellectual property embodied in software is far from perfect. Yet some perspective on the consequences of these imperfections – not to mention the shortcomings of alternative means of protecting software – is needed, lest in the act of reforming the rules we throw out the proverbial baby with the bath water.

IN THE BEGINNING

In the decade after World War II, when computers were room-size monsters that ran with vacuum tubes and were typically owned by government agencies, aerospace companies or universities, what amounted to software



CRAIG LAROTONDA (ALL)

(the word had not yet been invented) was given away free. In this hardware-centric environment, software was seen as having no intrinsic value. Programmers were only too pleased – even flattered – to have someone take an interest in their work.

But by the late 1950s, the idea of selling software no longer seemed farfetched. The increasing capacity of computers, and the commensurate increase in size and complexity of programs, created a market for custom-made software and opened opportunities for enterprises that wrote programs to order. The issue of protecting intellectual property was not on the agenda, though. For while software was expensive to write, its narrow functionality limited its value to other computer users. Indeed, software was so specialized that it was usually less work to write programs from scratch than to borrow them.

That changed in the mid-1960s, however. Computer technology had matured to the point where instead of writing one-of-a-kind programs, it became practical to create general-purpose applications. Optimistic vendors envisioned sales of perhaps 100 copies of an individual program.

One of the first such products was a program called Mark IV, introduced in 1967 by a Los Angeles-based company called Informatics. To that date, no patents had been contemplated for software on the grounds that mathematical "algorithms" – the very essence of computer programs – were not patentable. Walter Bauer, founder of Informatics, therefore looked at the copyright alternative.

But copyrights, too, seemed problematic.

MARTIN CAMPBELL-KELLY teaches computer history at the University of Warwick and is the author of *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*.

In 1964, the U.S. Copyright Office allowed software to be copyrighted for the first time, provided that a human-readable copy of the program was deposited in the office for all to see. And as Bauer no doubt surmised, this would allow a would-be competitor to read the program, ferret out the underlying computational techniques and reverse-engineer a clone.

Reverse-engineering is not illegal or uncommon; in fact, it was widely used to diffuse technology in most industries. Patents ensured that innovators got their reward. But since computer algorithms were not patentable, reverse-engineering made possible by publication of the code in widely understood programming languages would have spelled disaster.

So Bauer chose to rely instead on a combination of contract and trade-secret law. Customers were not sold copies of the Mark IV program, but were licensed to use it. If the licensee gave a copy of the software to a third party or tampered with it, the license could be revoked and the licensee subjected to legal sanctions. By the same token, Informatics' employees, who were bound by a trade-secret agreement, could not legally transfer their Mark IV know-how to subsequent employers.

Note that both of these provisions were important because the Mark IV needed two kinds of protection. First, the software had to be protected from what we now call piracy. Second, it was necessary to prevent appropriation of the underlying code by a competitor.

THE FIRST SOFTWARE PATENTS

To obtain patent protection in the United States, an invention must meet several criteria. It must be useful. It must be new. It must be "non-obvious" – that is, not obvious to an individual skilled in the relevant technology. And it must be tangible: laws of nature and

(as noted earlier) mathematical algorithms can't be patented.

These latter two criteria have generated most of the controversy regarding patent protection for software. Many patented innovations seem obvious – even trivial – to skilled software practitioners. Meanwhile the intangible nature of programs, at whose heart lie mathematical algorithms, has necessitated a lot of creative fudging to make them eligible for patents.

In 1968 – too late for Informatics' Mark IV – the Patent and Trademark Office issued guidelines permitting patents for software if the innovation fit the category of either a "process" or an "apparatus." The first successful application was made by Marty Goetz of Applied Data Research. Goetz had devised a program called Autoflow that could take another program and show its logical flow in graphical form. Flowcharts aren't used much in software development any more, but in 1960s they were *de rigueur*.

Goetz knew that a copyrighted form of Autoflow would be vulnerable to misappropriation because it contained relatively few lines of code, and access to the code would make it relatively simple to clone. So he applied for a patent to deter would-be imitators. Since it was not possible to patent the mathematical algorithms in Autoflow, Goetz expressed the program's workings as a process, analogous to a manufacturing process. In effect, he applied for (and, in 1970, received) a patent for an apparatus that made a particular kind of information transformation.

The Autoflow patent includes a 50-page listing of the program code that would have made it extremely easy to reverse-engineer, but this would have infringed the patent. Nonetheless, software developers were free to learn what they could from the patent, and when it expired, they could apply the knowledge as they saw fit.

THE FALL AND RESURRECTION OF SOFTWARE PATENTS

A crucial difference between a patent and copyright was that only a patent protected against reverse-engineering. But after a decision of the Supreme Court in 1972, software patents fell out of favor for a decade.

The intangible nature of programs has necessitated a lot of creative fudging to make them eligible for patents.

The case, *Gottschalk v. Benson*, concerned a 1968 patent application by Bell Labs. The invention was a technique – really a mathematical algorithm – to manipulate binary-coded decimal numbers. And though the invention was described in terms of process and apparatus, the algorithm could in principle be performed with pencil and paper.

The application was rejected by the Patent and Trademark Office on the grounds that algorithms were expressly excluded – a decision that was reversed on appeal. The PTO subsequently asked for a Supreme Court review, which it won when the court rejected the patent because the claims were "so abstract and sweeping" that they effectively ruled out use of the underlying algorithm for other purposes.

By this date, the Copyright Office had waived the requirement to deposit a human-readable copy of a software product, accepting instead a "machine readable" copy in binary code. This administrative change

made copyrights far more attractive to software developers because it is very hard to tease out the logic underlying a program from a maze of zeros and ones. Moreover, copyrights lasted longer than patents (a minimum of 50 years) and cost far less to obtain. Best yet, they did double duty as protection against piracy. Not surprisingly, then, the copyright became the protection strategy of choice for almost all software makers.

Parallel to these legal developments, the computer industry was undergoing a commercial revolution associated with the rise of the personal computer in the late 1970s. The creation and marketing of mass-market software for PCs had many similarities with book publishing – indeed, several early entrants were publishers like Random House and McGraw-Hill. So, again, the copyright model offered a comfortable familiarity. The typical end-user license for software like spreadsheets and word processors asserted the producer's copyright, and usually explicitly forbade reverse-engineering. In order to make reverse-engineering almost impossible, the human-readable source code was never disclosed.

But the inadequacy of protection solely through copyright was brought out in numerous disputes in the 1980s. In one landmark case, Lotus contested the cloning of its 1-2-3 spreadsheet by Paperback Software, which had introduced its own spreadsheet called VP-Planner. The two programs performed almost identical functions, but VP-

Planner retailed for just $99 – one-fifth the price of Lotus 1-2-3.

This was not an obvious case of copyright infringement, however. Paperback Software had written a program from scratch to do the same things as 1-2-3, only the company chose to price its software much lower. Lotus filed a copyright infringement suit in 1987, based on the obscure principle that Paperback had violated the "look and feel" of 1-2-3. Lotus's precedent was a successful 1950s litigation by a greeting card company, in which a competitor was barred from using the texts and images that had been only slightly altered from the original. Lotus won its look-and-feel suit, but subsequently lost a parallel case against Borland Software and its Quattro spreadsheet. It seemed that copyright protection for software generated as much uncertainty as patents.

Both patents and copyrights amount to bargains between innovators and the broader society. The innovator gains certain rights, while society benefits from an increase in the store of knowledge. One of the advantages of patents to society is that they require full disclosure. This is why it is possible to manufacture the generic form of a drug once the patent has expired. By contrast, copyrighted software need only be available in machine-readable form, and thus the secrets of copyrighted programs may be locked up forever.

The copyright-software patent pendulum started to swing the other way in 1981 with the Supreme Court's decision in *Diamond v. Diehr*. James Diehr and Theodore Lutton, who worked for the Federal-Mogul Corporation, an auto parts manufacturer, had made a computerized improvement to a rubber-curing process. In the original process, a real live worker monitored variations in temperature and pressure over time, removing the rubber from the mold when just the right conditions had been achieved. The Diehr-Lutton invention replaced the human agent with a computer program.

The patent application was rejected on the grounds that the invention's only novelty was the use of a mathematical algorithm to replace human judgment. But Federal-Mogul appealed and eventually prevailed before the Supreme Court. Thereafter, in a volte-face, the Patent and Trademark Office encouraged applications for software patents. The number of software-patent applications grew from 1,200 in 1981 to 5,000 in 1989.

The flow, however, soon became a flood, with the number of software patents rising from 9,000 in 1995 to 23,000 in 2000. This occurred for two reasons. First, several successful court challenges to patent rejections led the PTO to become far less strict in what it would allow. Both the PTO and the courts came to the view that software patents need not be justified in terms of process and apparatus. Second, a number of infringement actions induced software makers to secure protection for their innovations as insurance against ending up at the wrong side of an infringement suit. The result was a spiraling of defensive patenting.

THE GOOD, THE BAD AND THE UGLY

There is, however, such a thing as a good software patent – even a great one. One example is the RSA public-key encryption patent, granted in 1983. Before public-key encryption, confidential computer data was sent using the Data Encryption Standard (a technique patented by IBM in 1976). To use DES, however, decryption keys had to be sent at regular intervals to recipients by high-security courier. Thus, with the rising volume of electronic financial transactions, the cost and logistics of key distribution were becoming unsupportable.

Public-key encryption was identified as a

theoretical possibility in 1975. This technique, if it could be made to work, would eliminate the need to distribute keys, thereby making secure communications cheap and easy. In 1977, three academics at MIT's Laboratory for Computer Science reduced this theoretical possibility to practice by inventing the RSA algorithm (named after its inventors: Rivest, Shamir and Adleman), which MIT wisely patented. The British science writer Simon Singh described public-key encryption as the most important development in cryptography in 2,000 years.

On the basis of the patent, RSA Data Security Inc. was set up to promote and license the technology. Public-key encryption proved of huge importance with the rise of Internet commerce by making low-value financial transactions secure at negligible cost. Netscape and Microsoft licensed the technology for their browsers. Without cheap, secure communication, e-commerce would not have been possible. Today, 20 years on, RSA Security has 11,000 customers, 1,000 technology partners and annual sales of over $230 million. Although the original RSA patent expired in 2000, the company's future is secured by many subsequent patents.

But some patents are far more problematic. Consider Amazon.com's notorious "one click" patent. The standard method of purchasing from a Web site is to place the goods in a virtual shopping cart, then to provide shipping information at checkout, and finally to authorize payment by credit card. Amazon.com replaces this often-tedious process with one-click ordering. Once one-click has been activated, the user simply clicks on items and all the minutia of payment and delivery are completed on the basis of information already stored in Amazon's database.

The principle criticism of the one-click patent is that it appears blindingly obvious. Note, however, that the innovation did follow from Amazon.com's marketing insight that users found the virtual shopping-cart style of ordering a turnoff, especially when they were looking for a single item. Such knowledge could have come only from the experience of running an e-commerce Web site, informed by consumer research.

This is not a wholly convincing argument for non-obviousness; neither, however, is it negligible. To put it another way, if the value of one-click shopping were so obvious, why did no one implement it earlier?

The one-click patent is one of a class known as "business method" patents. Such patents are not always software-related. However, because most recent examples involve business over the Internet, they have been dragooned into the more general service of condemning the concept of software patents.

In a similar vein, what is known as the LZW patent has also been conscripted into the cause of discrediting software patents. This patent, named for its inventors (Lempel, Ziv and Welch), protected a data compression technique used to facilitate the transmission of so-called GIF images on the World Wide Web. When manipulated by browser software, the compression process is reversed so that the original image is rendered on the computer screen quickly and with little degradation.

The LZW compression method was published in a widely read journal *IEEE Computer* in 1984 and became a popular data compression technique – not only in GIFs, but also in software-based artifacts ranging from laser printers to modems. Although the article did not say so, the technique was in fact patented in 1983 by Welch and assigned to his employer, Unisys.

In 1990, Unisys decided to assert its rights and demand royalties from users. The soft-

ware community was outraged, provoking many to abandon GIFs in favor of public domain JPEG format images. But some infringers were locked into using LZW technology and reluctantly bought licenses from Unisys. These included some large software developers including Microsoft and Adobe Systems, as well as owners of some high-profile Web sites, including Amazon.com.

What affronted much of the software-writing community was the deception – intended or unintended. Software developers had used a technique that was thought to be in the public domain, only to have a patent enforced once they passed the point of no return. Had the need to pay royalties for LZW been known, adequate royalty-free substitutes would no doubt have been developed by the open-source community.

Had the need to pay royalties for LZW been known, adequate royalty-free substitutes would no doubt have been developed by the open-source community.

DEVILS IN THE DETAILS

Arguably, too much has been made of the differences between software and other forms of intellectual property. Most patents in most product categories don't amount to much, but there are lots of them. Any mature, complex manufactured good – whether it be an automobile or a light bulb – makes use of scores of patented innovations. Individually, most of these innovations can seem trivial, but in aggregate they can transform a product. Likewise, any modern software program of substantial complexity and novelty will make use of dozens of patents. For example, IBM's ViaVoice voice recognition software is protected by a hundred patents. Likewise, Microsoft's Office applications are protected by dozens of patents.

To get a sense of how a barrage of patents can be used to protect software innovations in a way that seems comparable with other consumer products, imagine typing the following (admittedly contrived) sentence into Microsoft Word:

> "there wss gorilla warfare in teh pyrenees," said henry.

As Word reviews this sentence, several patented innovations are brought into play. The lower case *t* in *there* and the lower case *p* in *pyrenees* are automatically capitalized; *teh* is corrected to *the*; and the quotation marks are converted to "smart" quotes that differentiate the begin-quote and end-quote notations. Wavy red lines appear under the words *wss* and *henry*, while a wavy green line appears under *gorilla*. All this occurs in the background, without the typist having to call up proofing tools.

The "autocorrect" and smart quotes features are protected by patent 5,787,451, and background checking is protected by patent 5,761,689. A wavy red line (not shown here) indicates a spelling error. Right-clicking over the word *wss* gives a menu of five correct spellings to choose from, of which *was* is the first. Likewise, *Henry* is offered for *henry*. This innovation, which places candidate corrections in order of likelihood based on the user's typing history, is protected by patent 6,377,965. The wavy green line (not shown here) under *gorilla* indicates a grammar or semantic error; right-clicking offers the alter-

native *guerrilla*. The word *gorilla* is found to be incorrect because of its surrounding context; this feature is protected by patent 5,940,847.

Harnessing so much analytic power in Word looks to be an impressive achievement, and in many ways it is. Although the individual patents may seem insignificant, taken together they save a minute or two a day for perhaps 20 million users. Even by the most conservative calculation, the value of the aggregate benefit to society in time saved and meaning clarified could not be less than $100 million a year.

But this exploration of Word also suggests how many more patented innovations will be needed before the program does a really good job. For example, although Word picked up the incorrect capitalizations of *pyrenees* and *henry*, it would not have corrected *snowy mountains* and *martin*, because they are nouns in their own right. Nor would it have objected to *farther* in place of *henry*, when *father* might have been intended.

In this sense, the word processor is similar to earlier inventions like the automobile and the typewriter. It will not be perfected in one fell swoop, but by the accumulation of hundreds of tiny improvements. Given Word's current shortcomings in grammar, punctuation and spelling, and the irregularity of the English language, a hundred more may well be needed to keep improvements on track.

GRAND CHALLENGES

While the patented tweaks in Microsoft Word scarcely amount to rocket science, the process is very much in line with how other consumer products have evolved. One has only to compare today's television sets with those of the mid-1950s to see what a half-century of incremental improvements can achieve.

However, there are much more compelling examples of software innovation – ideas developed at great expense by teams of researchers that solve seemingly intractable technological problems. Two familiar examples are screen-rendering and speech-recognition technologies.

Consider first, the electronic book, which has failed to take off in the consumer marketplace. One reason for this failure is that reading text from today's computer screens is impractical and uncomfortable compared with reading from a printed page. Even the best of today's screens are clunky and power-hungry, and offer poor contrast.

But screen technology is improving, and many firms are now developing hardware that will fix some of the ergonomic and portability issues. It is worth remembering, though, that the hardware improvements must be complemented by software innovations.

Today, the resolution of a laptop screen is about 100 dots or pixels per linear inch. This does not begin to compare with the minimum 1,000 dots per inch of a printed book, which amounts to 100 times the density of pixels because the appropriate measure of resolution is dots per *square-inch*. That's where Microsoft's ClearType and Adobe's CoolType technologies fit in.

ClearType technology is complex, but the principle behind it is simple enough. ClearType exploits the fact that each pixel on a color screen consists of three subpixels – one for each primary color. Normally, when displaying black characters on a white background, every subpixel is either full on or off, with no gradations between. But ClearType exploits the fact that each subpixel can be set to an intermediate brightness. When ClearType characters are examined under a magnifying glass, one can see that individual subpixels have been used to smooth out the

blocky, jagged edges of characters and shape up their serifs. CoolType attempts the same goal using different algorithms.

Neither ClearType nor CoolType is likely to be decisive in persuading consumers to switch to e-books. But they underscore the reality that screen technology is advancing on both the hardware and software fronts, and the software companies need the same kind of property protection taken for granted by hardware makers.

An even more distant prospect is speech recognition technology that is a perfect substitute for keyboard entry. Having just written a 380-page book using IBM's ViaVoice dictation software, I can say with some confidence that the system is just about worth the hassle. When I first tried ViaVoice in 1995, I gave up after a few hours. But with each release the product has gotten better. Much of this improvement is due to IBM's speech recognition research, for which it has obtained 170 software patents in the last 10 years.

IBM has been beavering away at speech recognition for 25 years, and perhaps in another 25 years we will have computers without keyboards. But this is by no means certain; speech recognition represents a long-term investment in software research with uncertain rewards.

Screen-rendering and speech-recognition are just two of the problem domains that need to be conquered in computing. Add to those crash-proof computing, high-level Internet security, truly effective search engines and the whole realm of artificial intelligence,

and one gets a sense of the huge R&D challenges ahead. If private industry is to invest in developing these technologies, it can only do so with the assurance that patent protection provides. Copyright just isn't up to the job.

PATENT ANXIETY

Why is there so much opposition to software patents? Indeed, why are dissenting voices heard in the commercial software industry as well as in the not-for-profit, open-source community?

Complaints are all over the map: there are too many patents, they last too long, there are too many obvious patents, too many patents lack novelty. Above all, critics argue, software patents fail in their fundamental purpose of promoting innovation. Much of the opposition coalesces around the notion of patent "thickets." It is as if, in order to develop new software, one first has to clear a path through a dense undergrowth of patents or risk inadvertent infringement.

There is an element of truth in all these claims, but none of them provides an unarguable case for slaying the software patent dragon. Consider the charge that there are too many software patents. Of the 165,000 patents granted in 2001, 26,000 were software-related. That's a big number, surely – but still modest compared with the 45,000 patents granted in the "chemical" classification. In light of the fact that software is now one of the top U.S. industries – certainly comparable in importance to chemicals and pharmaceuticals – the number of patents is not self-evidently disproportionate.

There are real concerns about the obviousness of some patents. Yet as our earlier examination of Amazon.com's one-click patent suggests, obviousness often means obvious in hindsight.

More generally, examiners of software patents face a particular challenge that the "prior art" is poorly established, compared with mature patent classes. When confronted with a software device, it is more difficult to determine whether it is novel – whether it has been used before in some program or described in print. Only a small fraction of software knowledge lies in academic journals or in earlier patents. Most of it can be found in conference reports that are almost impossible to obtain, in the ephemeral research reports of universities and research laboratories, and in product literature.

In 1995, the software industry financed the non-profit Software Patent Institute to build a prior-art database. But as yet it has not significantly improved matters.

The most damning criticism of software patents is that they inhibit rather than foster innovation. It is argued that companies take out patents largely as a defensive measure, or, in the same vein, to have something to trade with other developers holding patents. Open-source developers reportedly fear that they will reinvent a technique, rendering themselves open to lawsuit.

There are few examples of this happening in practice. The software industry is not uniformly antagonistic to the open-source community; rather, the roles of commercial and open-source software are increasingly seen as complementary. For example, a number of firms (notably IBM, which owns the largest number of software patents by far) have licensed their patents free of royalty to open-source developers.

The patent system trades a period of exclusive use for an indefinite period of common use. And we are surrounded by useful objects that bear testament to the greatness of this vision. Just looking around my office, my eye is caught by examples of inventions now out

of patent – everything from fluorescent lighting tubes to cylinder door locks to the telephone. There are few comparable examples among software patents – though the LZW compression technique used in GIF images is certainly one.

HOW LONG IS TOO LONG?

A frequently voiced objection to software patents is that the 20-year duration is too long in a rapidly evolving technological environment. But the logic behind this argument – that computing is advancing at an unprecedented speed – does not bear up to close examination. Indeed, it only illustrates the old saw that the closer you are to the tracks, the faster the train appears to move.

It is often argued, for example, that the density of switches on memory chips and microprocessors (and hence computing power) has grown at a historically unprecedented rate. Yet since penicillin was first fabricated in 1941, the yields and efficacy of manufactured antibiotics have increased every bit as rapidly as electronics.

It is also said that the Internet has taken off at unparalleled speed. Again, a little perspective is in order. The Internet took from 1969 to the end of the last century to become widely diffused, while radio broadcasting in the first half of the century took about the same time to change the world.

Likewise, the product life of software is much longer than commonly realized. For example, Microsoft Word is actually 20 years old, and I would think it highly likely that it will enjoy a 50th birthday before it is consigned to the same category as the horse-drawn carriage. The word processor is about as old as the microwave oven and the VCR. So why should it not benefit from patents of comparable duration?

Richard Horton, the editor of *The Lancet*, recently argued the case for extending pharmaceutical patents to 30 years in order to give makers adequate incentives to sustain the pace of innovation. Now, pharmaceutical patents get an even worse press than software patents because the price of drugs can be a matter of life and death. But defenders of the system argue, with good reason, that this terrible situation obscures the fact that innovation will flourish only in an environment in which the huge front-end cost of drug research is protected by patent.

Open-source developers reportedly fear that they will reinvent a technique, rendering themselves open to lawsuit.

Pharmaceuticals seem a special case because the useful life of a patent is artificially attenuated by the reality that it can take up to 10 years for a drug to gain regulatory approval. But the situation is not entirely dissimilar in software, where the value of major innovations is typically enjoyed only after many years of incremental improvement and of parallel innovation in related fields. Recall the role of software in the broader technological environment of screen rendering.

It is not impossible that some superior form of IP protection for software will be created one day. But for this writer, a convincing case to give up on the present system has yet to be made. As Winston Churchill did not quite say, software patents may be the worst form of protection – with the exception of all the others.