MATH YOU CAN'T USE

Patents, Copyright, and Software

null set:// if && document.forms[0].name !== "claimed"
elements [i].type ! = [blocked]
essage.value == "" { !== [revoked]
<dt>< face== [i] .name document.forms[0].name != "claimed"
message.value == ""{ !== [revoked] [revoked]
.elements.name[i].name != "blocked" >private property<
[revoked]
if (f.elements[i].name== ||""
for(f.elements[i].type != "claimed" && i ++) || i=f.length
&& i ++) i=f.elements[i].focus() [revoked]
<dt><font [REVOKE ALL PRIVILEGES] ****
>private< more information:</font> private:
<font face=document.elements.name[i]= "blocked" if (f.elements[i].name==
</font> <end> [REVOKE ALL PRIVILEGES] ****
for(f.elements.name[i].name != "blocked" && i ++)||i=f.length
].name != "blocked" && i ++)||i=f.length
if && document.forms[0].name !== "claimed" >private
elements.name[i].name != [blocked] [revoked]
>private property<
[revoked]

Ben Klemens

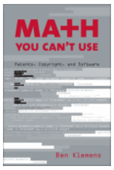
CHAPTER SIX

The Decentralized Software Market

The world of software engineering is in no way restricted to software companies. Beyond Microsoft or thousands of smaller software vendors, almost every corporation in the world keeps a stable of programmers in the basement to write little scripts that move the company's e-mail and make the "add to cart" button do what it should. I am a programmer because I write simulations and statistical analyses. Even you are a software programmer if you use the Record Macro feature of your spreadsheet or word processor.

The variety in types of software producers engenders two distinct methods of pricing software. One, *shrink-wrap pricing*, derives from more ephemeral markets: software is sold by the unit (packaged in shrink-wrapped boxes, for example) at a per unit cost. The other, *labor-oriented pricing*, follows from an hourly wage or an annual salary paid to people in the basement who write code. To give a music analogy: a band may record an album in the studio and then charge for each copy of the recording, or it can be paid for playing a live gig, and then audience members can bootleg the concert and listen at home for free. Both are viable means of making recordings for the public and money for the band.

The latest accounting from the Bureau of Economic Analysis divides the software market into three parts: retail, consultants, and in-house, which are evenly split in the U.S. economy. Of the $232.5 billion spent on software in 2002, 32.6 percent bought prepackaged programs, 36.4 per-



cent custom-built ones, and 31.0 percent software written in-house.[1] Since patent law is built around traditional products that are much more homogeneous, it is worth considering what will happen when the law for primarily product-oriented markets (such as drugs, machinery, and materials) is applied to a market that is one-third product oriented, one-third service oriented, and one-third a mix of the two.

Comparative Advantage and the Programmers in the Basement

A good company, according to the management self-help books, stays focused on its core functions. If a company is good at making orange juice, it does not digress into selling autos, even if the owner knows an awful lot about cars. But any business of more than a few people, regardless of its actual purpose, will need word processors, an accounting and inventory system attached to a database, a website, e-mail, and somewhere from one person to an entire department to take care of all that software.

By contrast, no companies have a drug manufacturer in the basement to make sure that the accounting department has all the Prozac it needs to function smoothly, and if the accountants find that off-the-shelf Prozac does not quite work, they cannot hire a chemist to hack, patch, or customize Prozac for the company's specialized needs. Yet no matter how much work is shifted to Microsoft, SAP, or other contractors, it will always come down to the in-house information technology (IT) department to make sure the company's software is installed and working properly. Although they are often invisible (until something breaks), the people in the basement are an integral part of the software industry.

The Communists Are Coming!

What happens to the software in the basement after it is written? Most software is so entirely location and task specific that it is used once and forgotten. Sometimes, however, it is so useful and new that the programmers in the basement form a company and start selling CDs—in fact, this is how a number of shrink-wrapped software products started out: for

1. By revenue; U.S. Department of Commerce, Bureau of Economic Analysis, "Recognition of Business and Government Expenditure for Software as Investment: Methodology and Quantitative Impacts, 1959–98" (www.bea.gov/bea/papers/software.pdf). Updated with 2002 data at www.bea.gov/bea/papers/table11.xls.

example, the SABRE flight reservation system (originally written at American Airlines, now owned by SABRE Holdings), the CADAM design program (originally from Lockheed, now owned by CADAM, Inc.), the Eudora e-mail client (written at and used by the University of Illinois, now owned by Qualcomm), or GAMS mathematical modeling software (written at the World Bank, now owned by GAMS Development).

With increasing frequency, software is also being given away to anyone who asks. Locking down a piece of software to license and sell it is just not worth the effort in the vast majority of cases—if a company has a comparative advantage in selling insurance or sofas, what business does it have in software consulting? It may have some great programmers in the basement, but hiring a sales team, getting the legal department up to speed on software licensing, and finding new ways to distribute software instead of sofas is a stretch far beyond the company's primary comparative advantage.

If a company is not hoping to make big profits from a piece of software, its best bet is to go to the other extreme and open the code base entirely, allowing for free and open collaboration. There are programmers in hundreds of basements who need a good database client. One writes the core of a database client and puts all the code out for inspection. Then another programmer, ensconced in another basement, finds that the code does what her company needs but has a few bugs, which she fixes. In another basement, another wage slave finds that the code works well, except it is missing support for BLOBs (binary large objects), so she adds that. The process continues, as everybody contributes the feature that makes the code perfect in their eyes, until—for the time needed to write a few functions—everyone has a full-featured and well-tested database.

Such collaboration in software dates back to when there were a handful of computers in the United States and a small community of programmers who knew how to work them. Since then collaboration has become infinitely easier thanks to the Internet.[2] Although some would claim that the collaborative system is the latest trend, shrink-wrapped software sold at unit cost is the new business model in this field.

It may sound like wishful thinking, but the collaborative method has produced some very heavy-duty software. As of September 2005,

2. A personal account of this history is given in the biography of Richard Stallman (Williams 2002), a vocal advocate of the collaborative method.

69.15 percent of web sites use Apache, a free program developed in about the same manner as just described.[3] As well as web pages, one's e-mail probably arrives via collaborative software (either Sendmail or IBM-sponsored Postfix), and all the computers involved found each other via the Internet addressing program that most servers use, Berkeley Internet Name Daemon (BIND), which is also free software. Such software goes by a variety of names, including free software, libre software, open-source software, or the catch-all FLOSS.[4]

A report assessing the popularity of FLOSS in three European countries has found some variation in its use: only 18 percent of establishments in Sweden use some sort of open-source software; 31 percent do so in the United Kingdom; and 44 percent do in Germany.[5] Sectors also vary greatly in this regard, with public sector organizations using more open-source software than those in the private sector.

Thirty-six percent of the companies surveyed agreed with the statement, "Our software developers are free to work on Open Source projects within their time at work," while 46 percent disagreed.[6] In other words, the plurality of companies insist that their employees' work remain the company's property, but a large percentage of for-profit enterprises allow some of their employees' work to be given away for free.

Open-source programmers are often characterized as hobbyists who are learning computer science or just having fun with pet projects of no

3. Netcraft 2005 Web Server Survey (news.netcraft.com/archives/web_server_survey.html). Microsoft's IIS comes in second, with a 20.36 percent share. A website is defined as one host name.

4. The naming of this type of software hints at some massive infighting among FLOSS advocates, even though they agree on virtually everything else. In a recent interview ("Thus Spake Stallman," *Slashdot*, May 1, 2000 [slashdot.org/interviews/00/05/01/1052216.shtml]), Richard Stallman, founder of the Free Software Foundation, takes pains to point out that "I am not affiliated with the Open Source Movement. I founded the Free Software Movement." He reserves especial vitriol for the writing of leading open-source advocate Eric S. Raymond, perhaps because Raymond has said of Stallman: "As an evangelist to the mainstream, he's been one fifteen-year long continuous disaster" (www.catb.org/~esr/writings/shut-up-and-show-them.html). The naming fight underscores the idea that the software should be free of licensing restrictions ("free as in speech") rather than simply free of cost ("free as in beer"). The term FLOSS is a pleasing compromise because it forms a common, albeit irrelevant, word from all of the options. Here, I refer to FLOSS as "open-source" software because I think it sounds nicer and also use the term "collaborative software" to refer to the means of producing software in a decentralized manner even when the output is not free or open.

5. International Institute of Infonomics (2004).

6. International Institute of Infonomics (2004, pt. I, sec. 4.1).

real significance. Although some of them would certainly meet that description, a reported 29 percent of Europe's open-source programmers are paid for developing free software at work, and 24 percent are not paid for doing so but do it on company time anyway.[7] Add to this the 17 percent of developers who are students, and only a few remain who are developing open-source software in their spare time.[8] The European Commission study states that "the development of Open Source/Free Software is not at all a matter of leisure 'work' at home. Ninety-five percent of the sample claim that they use OS/FS at work, school, or university."[9]

Making Money on Free Software

A number of little companies use free software to make money. For example, IBM sells mainframes, but if it can throw in free software that makes its mainframes powerful web and e-mail servers, then it can move more metal. In a similar vein, Sun gives away Java. Another name on the list of success stories is Red Hat, which provides consulting services for corporations and creates neat packages of free software for consumers. Hans Reiser, designer of the best UNIX file system (the reiserfs), sells features: he has a to-do list of a dozen features that he wants to implement in his file system, but when the president of MP3.com offered him tens of thousands of dollars to implement a feature necessary for MP3.com business, he quickly obliged. MP3.com saved millions of dollars by switching to free software using Reiser's free file system, and Reiser profited from what he would have done anyway.[10]

Collaborative software is clearly a threat to the shrink-wrapped software market, because, as the saying goes, it has to compete with free. But for the labor-oriented side of the market, the wealth of ready-to-download software merely creates new opportunities.

Free Software and Optimal Pricing

What does economic theory say about free software written by profit-maximizing firms? It says that this behavior is efficient. In a free and open market with many competitors, each unit of a good should be priced at the cost of producing that very unit (that is, the marginal cost). The first

7. Ghosh and others (2002).
8. International Institute of Infonomics (2004, pt. IV, sec. 2.3).
9. International Institute of Infonomics (2004, pt. IV, sec. 3.1).
10. Hans Reiser, speech at California Institute of Technology, November 14, 2002.

unit of software requires some amount of labor, and that needs to be compensated in full by paying the programmer a salary or wage. The second unit can be produced at basically no cost, since it only needs to be copied, so a zero price for the second unit and beyond is what the theory predicts and can be shown to be efficient.

Meanwhile, shrink-wrapped software is not priced by marginal cost, but closer to average cost—spend a million dollars making the first CD, then make ten thousand copies and charge $100 for each of them. Since vendors have a copyright on their work, their software will differ in enough ways from that of their competitors to allow them to charge well above marginal cost for their products.

In practice, of course, products are seldom priced at marginal cost. Few goods in this world are truly standardized, and even among those that are (corn of a certain grade, or government bond futures, for example), some units still sell at well above marginal cost. The amazing thing about open-source software, from the perspective of the theoretical economist, is that it *actually fits the theory*. Most markets experience problems that theory must ignore or explain away: inventories, shipping costs, transaction costs, massive up-front investments, and the risks those imply. Given such imperfections, it makes sense to correct them by imposing laws that would otherwise be suboptimal—a primary example being patent law, which solves the up-front investment problem. But collaborative software actually fits the models: transaction costs are nil, investment problems are solved without patents, and one can actually apply the theories that predict optimality without apology. For open-source software, patents solve an economic problem that had not existed to begin with.

Open Source and Patents

Not only do patents have no value or relevance to open-source software, but they have the potential to be a significant hindrance. By definition, open-source software lacks a centralized body through which to obtain patents, not to mention lawyers to defend against patent threats (although both IBM and Sun made limited pledges to support open-source authors in some patent-related issues, and even Lloyd's of London intends to sell liability insurance for servers running open-source software).[11] Adobe can sue Macromedia and vice versa, and both can afford

11. Other companies that have made patent pledges include Computer Associates, Nokia, Novell, and Red Hat. Robin Cover, ed., "Open Source Development Labs (OSDL)

to hire lawyers to keep them in business, but if anyone were to sue a collaborative project that does not have a patron backing it, the project would have no choice but to shut down.

The collaborative system depends on the source being open to all and making sure that everyone is free to modify the code. People who intend the code to be collaborative have an ingenious method of making it so: they copyright the code and claim complete control over its use. Then, in licensing out the code, they explicitly specify that users are free to redistribute or modify the code as they see fit, provided they do not impose their own restrictions. Although there are dozens of such contracts to choose from, the standard one delineating these rules is the GNU General Public License (GPL), where GNU stands for GNU's Not Unix.

Here is the key message from the GPL:[12] "This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License." How would this work if some portion of the code were patented? If the patent-holder charges a licensing fee, the software cannot be costless any more. If the software can be freely redistributed, the patent-holder must give up his or her right to limit distribution. Since the software can be reworked into other applications, the patent-holder even gives up the right to redistribution in a potentially wide range of applications. Clearly, any patent-holder who wants to retain any vestige of control would not consent to a patent being used in GPLed code.

The GPL explicitly acknowledges that if a claim is asserted for patented code in a project, the project must shut down as a public endeavor: "If a patent license would not permit royalty-free redistributon of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program." In short, patents and collaborative software cannot coexist, and if the two collide, patents win.

Collaborative software does have one advantage over patents. If a patent-holder threatens to shut down a collaborative project, the entire

Announces Patent Commons Project," Cover Pages, August 10, 2005 (http://xml.coverpages.org/ni2005-08-10-a.html). Gavin Clarke, "Lloyd's Taking on Open Source IP Risk," *The Register*, August 12, 2005 (www.theregister.co.uk/2005/08/12/opensource_indemnification); "IBM Statement of Non-Assertion of Named Patents against OSS" (www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf); Stephen Shankland, "Sun: Patent Use OK beyond Solaris Project," *Cnet News*, January 31, 2005 (news.com.com/Sun+Patent+use+OK+beyond+Solaris+project/2100-7344_3-5557658. html).

12. Version 2 (1991).

project may shift to finding prior art that would invalidate the patent. With hundreds of people from diverse parts of the computer science world all focusing on searching for prior art, the odds are very high that something will turn up. The Electronic Frontier Foundation has used this strategy to locate prior art about items in its list of the worst software patents.[13]

This approach still relies on a great deal of publicity, so if a specialized project receives a cease-and-desist letter or there is a flood of patent claims throughout the open-source community, the required critical mass of eyeballs required to find good prior art may not be reached. Every web designer in the world could probably contribute something to a prior art search pertaining to Amazon's infamous patent 5,960,411, on one-click purchasing. But if the GNU Scientific Library gets a takedown notice for violating one of the fast Fourier transform patents (see page 63), a far smaller population could come to the GSL's support.[14]

Even if a community of users find prior art immediately, it still needs to be ruled upon by the courts or the U.S. Patent and Trade Office (USPTO), which would take months (in Internet Time, several centuries) and may still fail on the details. Many users would be too risk-averse to wait for the ruling and would stop using the technology until clarity is restored.

Much has been made of security risks to the Internet and the potential havoc a terrorist could cause by a well-placed worm, virus, or technical glitch that might bring down large parts of the network. But here is the surest and simplest way to shut down the Internet: find a function or data structure in BIND or Apache that is under the scope of a patent, hire a lawyer, and start suing as many people as possible. For BIND especially, there are few alternatives, and switching is technically difficult—and who knows whether the alternatives are patent-free?

Microsoft has even thrown out a few warnings that such lawsuits are inevitable for users of open source software.[15] Fortunately, the company

13. The Electronic Frontier Foundation's Patent Busting Project, at eff.org/patent/.

14. This is not to say that the FFT patents are a special interest issue: not many people may know how to calculate FFTs, but most cell phones, DVD players, and cable boxes do. A disclaimer: when I wrote this sentence, I had in mind the maintainers of the GSL. But the description of how open-source software can benefit all involved earlier in this chapter was so persuasive that I have since initiated an open-source project based on the statistical functions I use in my own work (see apophenia.info). Therefore, the problem of patent exposure now applies to me directly.

15. John Lettice, "Use Linux and You *Will* Be Sued, Ballmer Tells Government," *The Register,* November 2004 (www.theregister.co.uk/2004/11/18/ballmer_linux_lawsuits/).

provides its own operating system and server software (IIS on Windows), which risk-averse companies can use to replace Apache on Linux, and provides indemnification protection by (and from) Microsoft's legal department.[16] Indeed, at least one case (*J2 Global Communications* v. *Mijanda, Inc.*) has cropped up over alleged patent infringement by open-source software used by the defendant.[17]

Even a single function could lead to a patent suit, so lawsuit-averse programmers had best purchase function libraries from vendors instead of writing their own and glue them together using a purchased copy of Microsoft's Visual Studio instead of a freely downloaded copy of the GNU Compiler Collection. Perhaps the best bet is to simply stop writing programs entirely and purchase all software from those centralized vendors who own the patent thickets that can provide indemnification.

At one time, the labor market and the shrink-wrap market were in a balanced relationship: software companies sold their goods to the labor-oriented side, which applied them to their projects, and everybody made money. But now that there is a well-established and tested mechanism to allow the labor-oriented side to incrementally build the operating system and desktop-level software that is the specialty of the shrink-wrap vendor, the goods-oriented side of the couple has been spurned. It is only natural that the goods-oriented side would use all of the weapons available to ensure that the labor side remains bound to the union.

Decentralization

Another way to cast the difference between the goods-oriented and labor-oriented market is to say that the labor-oriented market is massively decentralized. On one hand, there are only a few tractor companies, which maintain a full-time staff of the best and brightest. If those central repositories of mechanical knowledge are not well supported, the tractor arts cannot advance. There may be some inventive tinkerers cobbling

16. Ina Fried, "Microsoft to Back Customers in Infringement Cases," ZDNet, November 10, 2004 (http://news.zdnet.com/2100-3513_22-5445868.html).

17. Pamela Jones, "Patent Lawsuits That Involve FOSS," *Groklaw*, August 10, 2005 (www.groklaw.net/article.php?story=2005080914234645). Normally, using a patented device is *contributory infringement*, which can be prosecuted in a manner similar to direct infringement. But recall Judge Rich's opinion in *In re Alappat* (chapter 3): to load a program onto a computer is to build a new machine, meaning that Mijada is directly infringing the patent, even though its employees may not have written a single line of the open-source program that is the core of the infringement claims.

together contraptions outside of these companies, but the vast majority of tractor technology is developed and supported by tractor vendors. On the other hand, every basement of every corporation has its programmers, and they are producing fully operable software. If I want a program to do any given function, say, convert document formats or implement a database, dozens or even hundreds of viable options are at my disposal, only a fraction of which were written by people at software companies.

The abundance of languages and libraries helps: for any given task, there are so many tools already in existence that a designer can have a basic running program rather quickly. A wealth of database engines are lying around just for the taking, waiting to be built into larger devices; the same certainly could not be said of tractor engines.

The structure of software also makes decentralized programming easy. So long as he does not change the function's interface, a programmer can tweak, debug, and optimize the function implementation all he wants without affecting the other parts of the project that use the function. This means that after the overall high-level design is done, there is little or no benefit to having all of the programmers in one place. Of course, the fact that the product can be e-mailed instantaneously at zero cost helps as well.

I stress this decentralization because some pro-patent authors believe patent difficulties can be attributed entirely to the relationship between patents and open-source software. Since open-source advocates are mere hobbyists on the fringe, they reason, one can safely ignore them and focus policy on the vendors of software. As already mentioned, open-source software is neither written primarily by hobbyists nor produced on the fringe. Even so, the central problem is not about open source, but about centralized versus decentralized production. The best examples of decentralized production are indeed open source—the Linux kernel was written by 418 programmers from 35 countries, on every continent but Antarctica—but even the companies with a no-open-source policy have programmers in the basement working full time on code and software.[18]

If a technology needs a centralized group to help it advance, then it makes sense to design a mechanism to support those few specialized experts who push forward the frontiers. In such a field, the patent-thicket

18. Ilkka Tuomi, "Evolution of the Linux Credits File: Methodological Challenges and Reference Data for Open Source Research," June 2004 (www.firstmonday.dk/issues/issue9_6/tuomi/). Data based on kernel 2.4.25, released July 2002.

problem is not a problem because there are only a few actors in the business, so the transaction costs of negotiating exchanges are low.

But this story is entirely removed from the reality of software. A third of the industry consists of centralized organizations that only write software while the rest is largely a decentralized body of workers supporting themselves and their innovations through immediate, direct application rather than waiting to put out a product in the near future. As far as Coasian arguments about transaction costs are concerned, this is absolutely the worst case, since buyers and sellers are distributed across the planet. Because every patent is unique, there is no easy way to create a simple market to make patent trading cheap.

The rule that independent invention is not a defense in infringement claims makes sense in a centralized industry. Patents are public record, and it is reasonable to assume that every tractor manufacturer is exerting some effort to watch every other such manufacturer. In the decentralized software industry, this does not make any sense at all: should the sofa company spend time and effort on monitoring Microsoft and Novell's patent portfolio? Add in the software patent search problems from chapter 5, and the assumption that everyone has full knowledge of the patent playing field becomes still more tenuous.

In short, patents in a decentralized market are Coase's worst nightmare: every player needs to expend vast quantities searching for the owner of every part of every program, meaning transaction costs piled upon transaction costs. These costs will always exist in every field, but they are magnified in a dense, decentralized network of actors.

Centralizing the patent search process (by hiring centralized full-time patent search firms to support the decentralized programmers) will not help much: searchers will still have to check every computational nut and bolt, and owing to the joys of mathematical abstraction, hundreds of patents like the singular value decomposition patent apply to hundreds of different fields. Because any Turing machine can be applied to any effectively computable problem, computer science itself is a dense network of concepts, each one a step or two away from virtually every other. To do a proper search, then, one would have to check almost every prior use of a Turing machine: in all, 170,000 patents and counting.

As an aside, the political landscape of software is a manifestation of the *collective action problem*: a centralized group that stands to gain significantly from a policy will lobby more vehemently than a decentralized group of many people who all stand to lose from the policy, and so inefficient political decisions are often made to please the most concentrated and vocal interests.[19] At the height of the European debate, centralized producers with large patent portfolios such as Adobe, Cisco Systems, IBM, and Microsoft spent a great many resources on lobbying the EU's decisionmakers.

The patent problems discussed in this chapter are not about open source; they are about decentralization. Software design was decentralized before open source became mainstream, and at least a third of the market, including a large subportion that does not open its source, remains decentralized. Patents were not designed to cover goods produced by thousands of companies that do not even work in the industry in question. There are many reasons to believe that they are not as good a fit for a massively decentralized system as for traditional centralized systems of production.

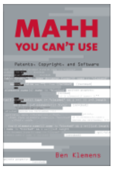
How Patents Affect the Bifurcated Market

Patents primarily benefit the authors of shrink-wrapped software. Returning to the music metaphor, the band that makes its money playing gigs needs no IP protection. If fans do not pay at the door, they will not hear the music. The band that focuses on CD sales depends heavily on IP protection, since copies of its CDs are near-perfect substitutes for the originals. Similarly, the provider of a software product needs to differentiate his from that of others in order to charge a unit price greater than the near-zero unit cost. Copyright is sufficient for this, but as discussed in chapter 5, patents as they exist today are so broad that a patent-owner can carve out sole ownership of a much larger part of the market than a copyright-owner could. Meanwhile, a strictly labor-oriented employee is more indifferent to IP protection: if the company does not pay at the door, then the programmer will withhold his or her labor—no IP required.

As already mentioned, patents make the most sense and provide the most economic benefit in a system built around a few centralized vendors of goods. Conversely, they make no sense at all in the context of a decentralized network of laborers—especially if everyone has already found incentive to innovate in the need to do his or her own job better.

In real life, of course, the class of programmers does not bifurcate into those who provide only shrink-wrapped software and those who provide

19. For the classic description of the problem, see Olson (1971).



only day-to-day labor, but includes people whose work falls all along the range. In the middle are a variety of consultants, who typically offer both software (either off the shelf or custom made) and implementation services. To the extent that they differentiate themselves through their unique software, patents may help; to the extent that they differentiate through high-quality labor, patents are irrelevant.

Just as some bar bands prefer to strictly control bootleg recordings and profit from their sale, labor-oriented providers may be able to profit from controlling the software they produce. In the context here, there is potential for a labor-oriented programmer to turn into a product-oriented programmer. However, recall the matter of comparative advantage: the sofa company is not oriented toward software sales or software patent licensing, and reorienting the business would be costly. Meanwhile, vendors of shrink-wrapped software know the software market and need to make little or no extension to the main business to apply for and license software patents. In short, software patents are designed for and can help shrink-wrap vendors but do nothing for the labor-oriented sector—except to the extent that labor-oriented workers are or could become shrink-wrap vendors as well.

The Future of Software

Allow me to make my predictions for the future of the software market. Computer services overall will continue to expand, while the market for shrink-wrapped software will become a smaller part of the equation, and the market for programming labor will expand. Of course, well-written shrink-wrapped software will always have a place in retail. Apple has shown that there is much to be said for having a professional design team working on the look and feel of a product, while authors of open-source and task-specific software are famous for poor graphic design. The open-source code base is constantly expanding, but that is no help here: although programs written in the mid-1980s often work perfectly today, goods that have not had a design overhaul since then look terrible to modern eyes.[20] Retail firms that put their effort into a good user interface will always have a market.

20. The open-source Athena widget set comes to mind.

Since consumers care much more about how their software looks and feels than database maintainers do, much of the demand is on the consumer side rather than the enterprise side. If nothing else, there is the games market: gamers have an insatiable desire for the faster, flashier, and newer items. However, games have no corporate clientele (at least not while the boss is watching), so they will never garner the hourly wage programmers, and open-source hobbyists have never been able to develop the critical mass of people necessary to put together the art, storyline, game play, and rendering needed to make a top-notch game.[21] Even so, gaming software is not just small change: sales in 2004 totaled $7.3 billion.[22]

There is more money yet in business software that nonprogrammers use (such as word processors and spreadsheets), which falls somewhere between the two extremes of beautiful games and ugly-but-efficient back ends. On the one hand, efficiency matters, but on the other, office workers are still human beings, and if they are going to spend a third of every twenty-four hours staring at a computer screen, it may as well look nice. In this range, things could go either way. To date, shrink-wrapped software has won out, because of aesthetic considerations and a strong focus on ease of initial use. But it does not have to be this way: the Department of Defense could hire programmers to add eye candy to OpenOffice.org (a collaboratively written office suite), and may still save money over licensing Microsoft Word.

Collaborative software is only getting better. OpenOffice.org is already more than sufficient for writing letters and balancing a home user's checkbook in a spreadsheet, and for all but the most demanding business uses. Even the French national police force uses this software on its 80,000 PCs.[23] The code base for OpenOffice.org will not disappear. If anything, the percentage of people who download free software that meets their

21. Sorry, Linux fans, but Tux Racer does not cut it.

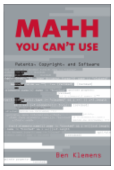
22. Entertainment Software Association, "Computer and Video Game Software Sales Reach Record $7.3 Billion in 2004," *Yahoo! Finance*, January 26, 2005 (biz.yahoo.com/bw/050126/265772_1.html). For comparison, Microsoft's 2004 annual report lists $36.8 billion in sales.

23. "Le Gendarme et OpenOffice," Toolinux, January 16, 2004 (www.toolinux.com/news/logiciels/le_gendarme_et_openoffice_ar5768.html). "French Police to Switch to OpenOffice," Heise Online, January 18, 2004 (www.heise.de/english/newsticker/news/55253).

needs is likely to expand in comparison with the percentage who spend a few hundred dollars on an office suite that does a little more and comes with animated characters. People prefer familiar, tested software, which currently means proprietary products, but companies such as Google and municipalities such as the State of Massachusetts are using open-source software, so the getting-acquainted phase has already begun. I expect that five years from now, collaborative software will be as familiar as the common brand names of today. This is a looming threat for the shrink-wrap market, but neither a plus nor a minus for the labor market.

On the consumer side, there is only so much that users need a word processor to do. Unless users can be persuaded to upgrade on a regular basis, software may be a one-time investment. For example, I have many friends who use Windows 95. They admit it with shame, since the name clearly indicates that the software is a decade old. Yet it still meets their needs and they see no value in the expense of upgrading. At the same time, things always break, so individuals and companies will need IT professionals on hand long after the software licenses have been paid for. I still get calls from my friends with Windows 95, since problems continue to crop up at a regular pace. By contrast, business computing needs are complex. Corporations are no longer satisfied with a straightforward personnel database—they want one that automatically makes hiring decisions the way the vice president would make them, that integrates seamlessly with the accounting database, and that has an interface on the company's website. None of these things can be pulled out of a box; a programmer who knows the company will have to be hired to implement them.

Authors of retail software can be located in Seattle, India, or anywhere in between. As noted earlier, any competent programmer can implement any sufficiently detailed interface design, although a consultant hired to design the interface for a company is very likely to be on site, getting the lay of the company's virtual land. With increasing outsourcing and offshoring (today's software market buzzwords), the number of domestic programmers writing shrink-wrapped software will decrease, but there will be less effect on the domestic programmers writing customized software. For all of these reasons, I foresee slower growth or even some contraction for shrink-wrapped software in the near future, whereas the market for custom programming labor will expand in close proportion to the increasing ubiquity and complexity of computing.



Back to Patents

Patents favor the shrink-wrap market, which is the segment of the market likely to experience a decline. In this context, software patent laws are among those that economists despise most: namely, laws that artificially prop up an industry in decline. Like a spurned lover, the centralized vendors will fight to keep their portion of the market. Stronger patent laws to bear down on decentralized labor are a primary weapon in the fight.

If the market does not stay as it is today but shifts further from the per unit model toward the labor model, it is hard to predict whether the total number of programmers will rise or fall. Certainly, the information technology sector as a whole is not likely to suffer. On a more abstract level, free software or task-specific software can be expected to add as much or more value than shrink-wrapped software, and authors in the software labor market are likely to match or outdo software vendors when it comes to innovativeness.

From the perspective of society, and even the software industry as a whole, there is no need to protect the shrink-wrap segment of the market, or to change the rules to favor it over the labor-oriented segment. Yet that is exactly what software patents do, at the cost of hundreds of millions of dollars wasted in litigation.