2
What Good Is Free Software?

James Bessen

When you use the World Wide Web, it is likely that the web pages are sent to you by software that was developed by unpaid volunteers. The majority of web servers on the public Internet use an open source software program called Apache.[1] This free product was developed by a loosely organized team of hundreds of programmers and thousands of other people reporting bugs or requesting enhancements, none of whom were paid by the Apache group for their efforts.[2] Yet Apache competes successfully with well-funded commercial products developed by Microsoft, Sun, and other companies.

This seems paradoxical in an age when conventional wisdom holds that markets driven purely by private interest best serve collective needs. But I argue here that the paradox has a straightforward explanation. Although some developers of open source software are indeed motivated by altruism, many are members of communities that benefit from the development of software along very flexible lines. In particular, firms driven by the conventional profit motive find open source software valuable because it allows them to meet their specific idiosyncratic needs—needs not easily met with standardized software products. I argue that for complex software products these unmet needs constitute a major source of demand, providing a robust long-term economic foundation for open source software.

In a sense, open source provision is an *extension* of the market, not an alternative. Private agents meet private needs. As I explain below, instead of providing software in exchange for money, open source developers provide software in exchange for a (sometimes informal) promise to improve the product and return the fruits of their invention to the community.

Government support is thus not necessary to sustain open source development. However, I argue that the U.S. government is nonetheless intervening in this market in a very different way, ironically sabotaging the otherwise healthy open source movement.

Over the last two decades, the courts have radically changed the legal protection of the ideas embodied in software, making it much easier to obtain patents, even for rather obvious ideas. The courts made these changes despite a high level of innovation in the software industry and without any evidence that these patents would improve (or had improved) the pace of innovation. In response, large firms have acquired thousands of patents in order to strategically block competitors. The resulting "patent thickets" threaten the ability of open source developers to improve software and may thus undermine this important source of innovation in the future.

This chapter offers two contributions. First, it presents an economic rationale for the participation of for-profit enterprises in open source development of complex products.[3] Much of the previous economics research on open source has focused on the participation of individual users who are also programmers (user-developers) with simple needs. For example, Josh Lerner and Jean Tirole attribute much individual motivation to reputation building, while Justin Johnson and Jennifer Kuan model individual user-developers with common needs but heterogeneous valuations and abilities.[4] Second, this chapter presents the argument that changes in patent regulation, in particular those encouraging the growth of patent thickets, threaten the development of open source software.

The discussion begins with a description of open source software and free software, followed by a comparison of these forms

to proprietary provision of software. In subsequent sections, I argue that open source does not generally require government financial support, and I briefly consider government procurement policies with this reality in mind. The chapter concludes with an analysis of the effects of patents on open source, with recommendations for changes in patent policy.

Free and Open Source Software

Since the early days of computing, users have shared computer code. Many important early programs, including many developed with government funding, were freely passed around. In the 1950s and 1960s, proprietary software consisted of limited applications that were almost entirely sold bundled with computer hardware. Little packaged software was sold until the 1970s, when IBM was challenged by private and government lawsuits to unbundle and when minicomputers came into wide use.[5]

In the mid-1980s a new, more formalized model for sharing software code emerged. The computer scientist Richard Stallman, concerned about limits on his ability to access, modify, and improve software, started the free software movement.[6] He developed the GNU General Public License (GPL) for software programs. Under the GPL the user obtains free access to the software code and agrees that any redistribution of the code will also be freely available, including any modifications the user makes to the code.

Free software gained momentum during the mid-1990s, with the emergence of the Internet. Developers such as Linus Torvalds, the initial creator of Linux, pioneered new organizational schemes that made it possible for hundreds of volunteer programmers to participate in joint software development over the Internet. Out of this broad participation arose the open source movement, which includes software developed under the GPL as well as other license agreements.

It is helpful to define some terms. First, note that there are two senses in which free software is free: it has zero direct cost

to the user, and it provides the freedom to modify the software. Stallman emphasizes the latter usage. Free software, he explains, is "free as in 'free speech,' not as in 'free beer.'"[7] This distinction is important for two reasons. First, free software is not at all the same as "freeware," which is zero-price software with closed source code that is often provided as a trial product. Second, it is highly misleading to view the main economic attribute of free software as its price. As is well known, the total cost of installing a software program includes many other costs; even with proprietary software, the price of the software is usually only a modest portion of the total user cost.[8] Also, as I explain below, large economic benefits arise from the freedom to modify the source code.

Open source software includes free software subject to the GPL, but it also includes other license agreements that permit access to the source code. Some of these license agreements permit the user to incorporate free code into proprietary, closed source products. For example, versions of open source Unix have been incorporated into closed source operating systems under non-GPL licenses.

Note, however, that for many products, public code is rarely included in proprietary products even when the license allows this. That is particularly true for dynamically improving products, in contrast to mature products such as standard Unix. First, open source developers are not motivated to improve software if they suspect it will be converted to a proprietary product. Second, and perhaps more important, creating proprietary software from open source code is often difficult because open source software typically changes rapidly. One of the key aspects of open source products such as Apache is that large numbers of modifications, improvements, and bug fixes are made and rapidly incorporated into new releases. Any private modifications would have to be continually re-integrated into new releases at significant cost. So although someone could legally use the Apache code to produce a customized closed source product, this has not been done. Firms offering proprietary web servers develop their own code from scratch.

Here I use the term "open source" to include free software distributed under the GPL as well as software distributed under other licenses.

To understand the nature of open source software, it is necessary to dispel some misunderstandings. Reading the business press, one might conclude that open source software is a marginal, transitory activity that lacks the solid economic rationale of proprietary software. Open source software is portrayed as the province of young idealists, graduate students, and teenage hackers rather than the serious business of corporate management information systems (MIS) departments with mission-critical computer systems to keep online. Others suggest open source is something of a fad.[9] Perhaps most important, open source programmers are presumed to be motivated by altruism rather than by traditional profit motives—as the *Economist* tells us, "for love, not money."[10]

This last point, in particular, reinforces the occupational prejudices of economists. They tend to think in terms of standardized commodities because they have had considerable success explaining markets for standardized proprietary commodities. They naturally view Microsoft Office as the ideal sort of software, while Microsoft is frequently viewed as the prototypical software enterprise. Furthermore, they have a lot of experience suggesting that traditional proprietary incentives are highly efficient means of providing standardized commodities.

Consequently, economists, if not outright skeptical about open source, tend to view it as a puzzle to be explained. Clearly, it is possible for a bunch of idealistic programmers to write a lot of code, but staying power requires more than just altruism.

The pundits said many of the same things about the PC and PC software twenty years ago. PC software was written by young idealists who supposedly weren't up to the task of writing serious corporate software. After all, a college dropout (Gates at Microsoft) and a Transcendental Meditation instructor (Kapor at Lotus) ran the leading PC software companies. But, of course, new software is usually written by people who do not have a large vested interest in old software.

In reality, open source software has become serious business. IBM is investing a billion dollars in open source projects, and many other large companies are joining in, too. By the same token, some open source products have achieved a high degree of success, such as the Apache web server, which holds a 64 percent share of active, publicly accessible web servers.[11] A study at the University of Wisconsin found open source UNIX operating systems were more reliable than more mature commercial products, and a study at Berkeley found superior debugging among open source software projects.[12] In addition, industry surveys have found a higher degree of customer satisfaction among open source users.[13] Note, too, that open source programmers are not primarily teenage hackers—a recent survey found that the average active participant had ten years of programming experience.[14]

It is true that most open source products are directed at technically sophisticated users, and many are not very "user friendly." Some people argue that this is no accident: open source coders are unlikely to develop programs for less sophisticated users.[15] However, the current technical bias may just be a result of the newness of the software. Early PC and minicomputer software was hardly user-friendly—Microsoft took over a decade to deliver products easily grasped by the untutored. And there are important signs of ongoing progress. Graphical user interfaces (from KDE and Gnome) for the GNU/Linux operating system are now widely used.[16] And according to some reviewers, the recently released open source Mozilla web browser can meet or beat Microsoft's Internet Explorer for usability.[17] Thus it is simply too soon to tell whether open source is in any fundamental way limited in its ability to address the needs of less technical users.[18]

Not only has open source software achieved some important successes, it also appears to be gaining momentum and improving rapidly. The number of developers registered at SourceForge, a popular website for open source participants, continues to grow rapidly and now exceeds 400,000.[19] The software they produce has grown in sophistication and features. For example, the Apache program was initially used for small, individual web servers. Large

websites that handled tens of thousands of users at once needed industrial strength "application servers," such as Weblogic from BEA. Now, however, open source products such as Tomcat and Jboss are moving into that territory.[20]

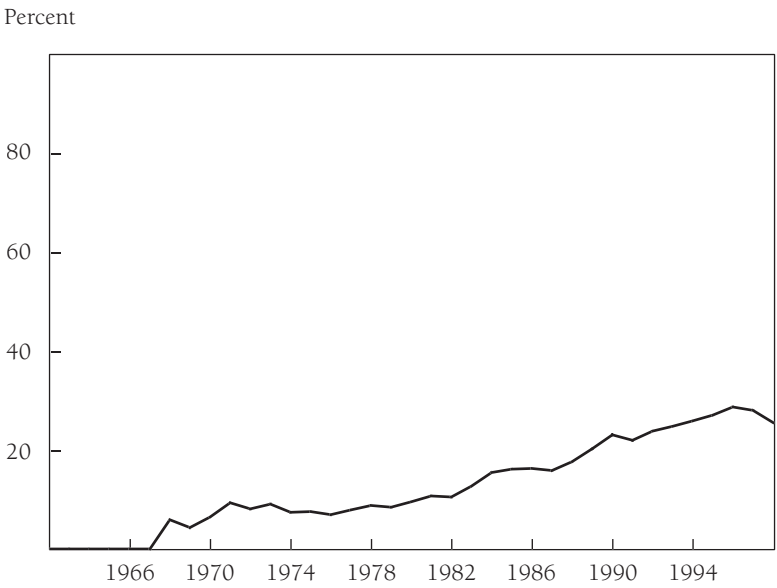
Finally, it is simply not true that most open source developers are solely motivated by altruism. A recent survey by the Boston Consulting Group found a wide variety of motives.[21] Some developers get involved with open source projects to learn cutting edge technology. For this reason, open source programs are now widely used in university computer science courses. Others seek a sense of community in participating in open source projects. Yet others hope to build their reputation through involvement that advances their careers.[22] Finally, in many cases, participation in open source development permits individuals and firms to obtain software that is customized to their particular needs.

All of these motives are important. I will focus on the last one—the ability to customize software for in-house needs—because I see this incentive as a critical reason why firms (and not just individual programmers) want to participate in open source development and why a growing share of software is likely to be developed with open source code. Indeed, as I elaborate below, the need for customized software may drive firms to open source development to improve the bottom line. Thus it is a false dichotomy to pose open source and proprietary development as a contrast between altruism and private incentives.

Software and Markets

A large economics literature convincingly argues that markets are efficient mechanisms for meeting many private needs. Markets can provide the strongest incentives for private agents to undertake the investments necessary to make the commodities that best meet the demand. For example, consider standardized commodities in large markets. It may be too expensive for each consumer to produce the commodity for his or her own needs.

Figure 2-1. Packaged Software Share of Investment, 1962–98



Source: Data from Robert Parker and Bruce Grimm, "Recognition of Business and Government Expenditures for Software as Investment: Methodology and Quantitative Impacts, 1959–98," report presented at the meeting of the Bureau of Economic Analysis, May 5, 2000 (www.bea.doc.gov/bea/papers/software.pdf [August 2002]).

The market serves to aggregate demand for a standardized product from many different consumers. Then a firm can make profits that are sufficient to cover production and development costs.

But most software is not a standardized commodity. Although Microsoft is often viewed as the prototypical software firm, most software is not packaged software. Indeed, as figure 2-1 illustrates, packaged software has never accounted for more than a third of software investment. The majority of software produced is either self-developed or custom. This is quite different from typical commodities, and for this reason, arguments grounded

in analysis of typical commodities need to be examined carefully. The ability to tailor programs to meet specific needs is an essential characteristic of software. This is what is "soft" in software, and this is why software-hardware systems are economically advantageous in so many applications.

The customizability of this good affects the nature of the product and the nature of the market. Standardized software packages are, at best, a compromise. Attempting to meet as large a set of individual consumer's needs as possible, standardized packages cram in as many features as is feasible. This is why successful products so often suffer from feature "bloat."[23] Standardized packages also frequently have "macro extensions"—tools included in word processor, spreadsheet, or other applications that allow consumers to customize the packages to a limited degree with a programming language. But the cost of debugging ultimately limits the ability of standardized products to meet highly disparate needs. As products accumulate features, their code becomes much more complex, and debugging costs grow exponentially.[24] Thus developers of standardized products face a trade-off between feature richness and product quality and reliability.[25] Restricting features limits the ability of standardized products to meet the needs of software consumers. Hence, packaged software represents only a minority of software investment.

Custom software and contract programming provide an alternative proprietary means for meeting the needs of individual consumers. But customization will not, in general, meet the needs of all consumers for two well-understood reasons. First, as anyone who has negotiated a custom programming contract knows, it is very difficult to specify the contract. This is because the only complete specification of all the software features and its behavior under all circumstances is the software code itself.

In effect, consumers cannot specify what they want the software to do in all circumstances until they actually have the software in hand. Any software contract is thus what economists call an incomplete contract. When contracts are incomplete, many consumers will not have their needs met by proprietary provid-

ers. Second, negotiations over custom contracts also suffer from asymmetric information.[26] That is, the developer does not know how much value the consumer places on the product and therefore does not know what to charge. With standardized commodities, market demand is revealed through many transactions. But with a custom product, developers cannot obtain this information without a costly bilateral bargaining process. As a result, a developer may not offer a custom contract, even though a contract on terms that are profitable would be accepted. Alternatively, the developer may overreach and ask for too much, leading the consumer to reject the offer.

Incomplete contracts and asymmetric information result in some degree of market failure.[27] That is, some consumers are not served even though their needs could be met with profitable contracts. Note, however, that such market imperfections cannot be corrected by direct government intervention—the government is in no better position to design or negotiate contracts than private software developers.

But open source development may finesse these problems by allowing consumers to customize products themselves. Open source means, roughly, that consumers are provided with 99 percent of their desired product in a form that allows some of them to tailor the remaining 1 percent to their own needs. When the consumer *is* the developer, problems of specification or valuation are easily resolved.

The open source developer makes the desired enhancements or fixes and then, under the most common licenses, submits them back to the open source project. They are then incorporated into future releases and improve the value of the product for other consumers who may have similar needs. This works for enhancements to product features, for entirely new features, and for bug fixes. Because so many active user-developers become involved, the quality, reliability, and overall value of the product can grow quite rapidly.

Moreover, many consumers of software who aren't developers have the same needs as some of the user-developers.[28] In these cases, the feature-rich products and the large variety of

customized add-on modules produced by open source user-developers benefit many nondevelopers as well.

The Apache web server illustrates the importance of customization and feature enhancement for open source users. In one survey regarding security features, 19 percent of the firms using Apache had modified the code to these ends, and another 33 percent customized the product by incorporating add-on security modules available from third parties.[29] Open source code facilitates the provision of add-on modules, and over 300 of these have been developed for Apache.[30] Many are quite popular: sixteen add-ons have at least 1 percent market share, and one (PHP) has 45 percent market share.[31] Moreover, many private enhancements are shared with the community and incorporated in new versions of the product. During the first three years of Apache, 388 developers contributed 6,092 enhancements and fixed 695 bugs.[32] This far exceeds the rate of feature enhancement for comparable commercial products.[33]

The breadth and dynamism of this participation demonstrate the degree to which open source software extends the market. Apache is used primarily by firms, not by individuals. So firms choose to customize their software and then choose to contribute these modifications back to the Apache group or to make them available as add-ons. Although personal motivations such as altruism, learning, community participation, and reputation all contribute to open source development, firms obtain the direct benefit of software tailored to their own needs. The many firms that customize Apache represent consumers whose needs are largely not met by proprietary products.[34] In addition, open source meets the needs of firms that cannot develop their own software but that have the same needs as some of the customizing firms. And it also serves those consumers who cannot afford to license proprietary products. Open source thus provides a means to extend the market. Although it does not involve the exchange of goods for a positive price, it is an exchange of a software product for a (sometimes informal) promise to return possible enhancements to the community. That promise comes true frequently enough to sustain participation in open source development.

Of course, proprietary software companies can achieve some degree of customization by permitting third parties to make add-on products. But the third-party community for open source can be much more robust than that for proprietary products. For example, in contrast to Apache's 300 add-ons, the third-party listing for Microsoft's comparable web server product, IIS, contains the names of just 11 partner companies.[35]

There are two reasons for this difference. First, open source third-party development is just a counterpart to the open source process of code development; third-party add-ons are just those enhancements that the developer or Apache Group chose not to include in the standard Apache product release. But the developers of these add-ons have the advantage of full access to the source code, documentation, associated tools, and the advice (through email newsgroups) of the development community. This makes third-party development highly dynamic. Microsoft tries to encourage third-party development, and it recently announced a "shared source" initiative that permits select third-party developers to see, but not to modify, Microsoft source code. Nevertheless, it appears difficult for Microsoft or other proprietary developers to duplicate the kind of dynamic community supporting some open source products.

Second, proprietary software companies deliberately limit the role of third-party developers, choosing to develop some erstwhile add-ons themselves and preventing some from being developed at all. Third-party developers depend on application program interfaces (APIs) to integrate add-on products. Proprietary developers only make limited APIs available, and these change over time. For example, companies such as Netscape and RealNetworks experienced difficulty with their Microsoft add-ons as Microsoft changed APIs. Indeed, opening the APIs to third-party companies has been an issue in the Microsoft antitrust suit.

On the other hand, open source development is sometimes limited for another reason. The success of open source projects depends on a sufficient accumulation of code to make it worthwhile for developers to begin the customization-modification-improvement cycle. To some degree, open source projects face a

chicken-and-egg problem. Until the project reaches a critical mass, it may be unattractive to many developers; developers may choose to wait for others to serve as pioneers. This is a version of what economists call the "free rider" problem.[36] Thus some open source projects fail to get off the ground where proprietary products can succeed.[37] However, once open source products do reach critical mass, they may very well prove superior to proprietary products at providing customer solutions.[38] And, fortunately, many of the alternative motives for programmers propel them forward.

In a way, standardized products and open source products are mirror images. Standardized products succeed by finding a common denominator of features that meet a portion of the needs of a large number of consumers. The market serves to aggregate demand, but the products do not satisfy consumers' specialized needs. On the other hand, open source aggregates *supply*: many different consumer-developers contribute modifications, enhancements, and fixes to meet a wide variety of different needs. However, to do so, open source projects need to begin with a common denominator of code that meets basic needs.

In summary, open source software meets a set of private economic needs that are not well served by proprietary software. The *Economist*'s false dichotomy posing "love" against "money " misses the reality. A profit-driven firm that is unable to purchase packaged software that meets its critical business needs may well find open source a rational and highly effective solution. For this reason, I believe that open source software will continue to thrive and become progressively more a part of the mainstream.

Open Source Software and Government Subsidy

Open source software allows consumers to meet needs that are not met by proprietary software. In a narrow technical sense, then, the widespread use of open source suggests opportunities created by market failure—that is, some socially beneficial transactions do not occur in traditionally organized markets. But this does not imply that government intervention is appropriate to

correct these failures. In fact, open source software is itself a private means of remedying some of these market imperfections.

In a broader sense, open source can be viewed as an extension of the market, a voluntary exchange between private parties. As such, direct government involvement is not needed absent evidence of other market failure. Open source has clearly flourished so far with little government support, and as the aforementioned SourceForge statistics suggest, open source is continuing to grow without much government support.[39]

One area in which government has a direct financial impact on both open source and proprietary software is in procurement. Various governments around the world appear to be tilting one way or the other. China recently rejected proprietary solutions in favor of open source products for a variety of projects.[40] In part, the Chinese seem motivated by a wish to cultivate a domestic software industry. Last year France announced support of open standards and recommended that government agencies use open source products, rationalizing the initiative as part of a plan to encourage the growth of small and medium-size software companies.[41] On the other hand, many existing procurement policies work against open source. For example, defense procurement in the United States requires security certification from the National Security Agency (NSA). Yet recent changes in the NSA certification process require the software vendor to pay the costs at commercial testing labs.[42] This effectively excludes much open source software. In addition, there appear to be some efforts to discourage or even to *ban* open source products from some defense procurement.[43]

Governments are clearly formulating different policies with different effects. The considerations are necessarily complex. However, I can suggest some simple guidelines. First, it makes no sense to have procurement policies that discourage consideration of open source products.[44] Second, most of the costs and benefits of any software purchase arise from the direct costs and benefits of the specific application. For this reason, products should largely be considered on their merits for the project at hand. Note, however, that when future modifications are important, open

source may provide added flexibility, and thus future costs and benefits should be factored into the calculation. Also, whether the software is open source or not, systems should support free, open standards, so that future users need not be "locked in" to a particular product in order to access the data.[45]

Finally, it is true that there might be significant positive externalities associated with open source—that is, benefits that accrue to parties other than the decisionmakers. The Chinese and French governments appear to take this into account, viewing open source as part of a national "industrial policy" to promote competitiveness in the software industry.

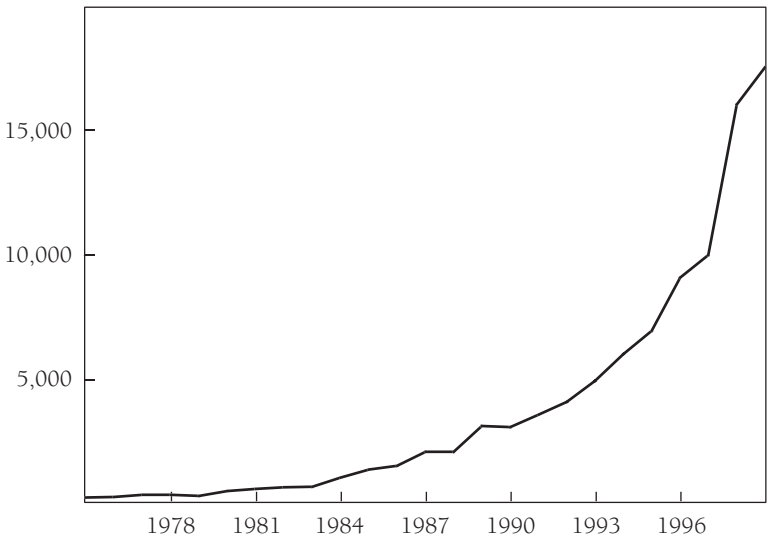
But as earlier debates over industrial policy suggest, it is often very difficult to measure the potential benefits.[46] Indeed, it is hard to see what government procurement provides small software firms that they do not obtain from their own use of open source software. And while government support might help new open source projects get off the ground, many proposed projects are not socially beneficial, and government possesses no better knowledge than private parties about which proposed projects address unmet private needs.[47] Perhaps positive externalities can be demonstrated in certain cases. Nevertheless, the evidence to date does not warrant a blanket preference for one form of software provision over another.

Patent Thickets and the Future of Open Source Software

In another area, however, government policy may have a deeply chilling effect on open source development. Intellectual property rights regulate the exchange of software for both proprietary and open source products. Dramatic changes in patent regulation, largely initiated by the courts with little industry or legislative input, pose a significant threat to the future health of software and to open source software in particular.

Prior to the mid-1980s, trade secrecy law and licensing contracts protected techniques used in developing software. Copyright law protected against piracy. With these protections, the software industry was highly innovative, and in fact, most of today's leading software companies grew up in this environment

Figure 2-2. Software Patents Granted, 1975–99



Source: Gregory Aharonian, PATNEWS, Internet Patent News Service (www.bustpatents.com [August 2002]).

lacking patent protection. The only apparent shortcoming of the system was its failure to provide employment for intellectual property attorneys.

That problem was fixed during the 1980s, when the courts engaged in a bold social experiment, extending patent protection to software. At the same time, they made other changes that resulted in a lower standard for nonobviousness and a greater presumption that any patent granted by the patent office was valid.[48]

The result was a dramatic patent "land grab" (see figure 2-2). The number of software patents soared to almost 20,000 per year, and a total of nearly 100,000 have accumulated to date. The largest share of these patents was obtained not by software companies, but by hardware companies building large portfolios—so-called patent thickets. Many leading software companies expressed the view that patents were undesirable, although

many of these same companies have since been forced to acquire patent portfolios for defensive purposes.[49] Predictably, the number of patent lawsuits has also begun to soar.[50]

Open source developers simply cannot afford to play the strategic games in which firms with large portfolios of patents routinely engage. In a survey conducted by researchers at Carnegie Mellon University, 82 percent of the respondents said they patented product innovations to block competitors.[51] In other words, an important use of patents is to *prevent* innovation by competitors. Also, 59 percent said they obtained patents to prevent lawsuits, and 47 percent obtained them as aids to negotiations. That is, firms use patents, especially portfolios of patents, as bargaining chips. Many of these patents are never used commercially.[52] This behavior arises when technologies are complex, cumulative, and overlapping so that any single product may potentially infringe hundreds or thousands of patents. That's why semiconductor firms regularly engage in strategic cross-licensing of whole portfolios of patents.[53]

Software is very similar to semiconductors in that any single product contains hundreds or thousands of "inventions." Because large firms can acquire thousands of patents—a task made much easier thanks to the lowering of standards (see below)—they are beginning to play the same strategic games, and software cross-licensing agreements are the result.

Open source developers are well aware that they cannot easily play these games.[54] Individual developers do not have the resources to acquire sizeable patent portfolios. The process of negotiating and litigating portfolios is even more expensive. As Bill Gates wrote in a memo, "A future start-up with no patents of its own will be forced to pay whatever price the giants choose to impose. That price might be high: Established companies have an interest in excluding future competitors."[55]

Firms that use or promote open source have resources, and many also have large patent portfolios. Some, such as IBM, license their patents royalty-free to open source developers.[56] Other developers talk of responding to the emerging threat by building common defensive patent pools.[57] But to date, no significant patent

Figure 2-3. Example of a Trivial Patent

United States Patent *5,851,117*
Alsheimer, et al. **December 22, 1998**

Building block training systems and training methods

Abstract

A building block training system and method of training of cleaners of facilities to be used on the job which utilizes a plurality of pictorial displays showing a specific set of steps to accomplish a cleaning operation in an efficient safe manner, e.g., dusting or vacuuming of a facility as well as a plurality of pictorial displays as to what must not be missed and must be avoided in performing the cleaning operation.

Source: U.S. Patent and Trademark Office (www.uspto.gov/main/patents.htm [August 2002]).

pool or other organization has emerged that can protect open source products from such strategic competition, and the coordination and transaction costs of building such a solution seem quite high.

To date, only a few open source developers have been directly embroiled in patent infringement controversies.[58] Similarly, litigation activity and cross-licensing among proprietary software developers has only recently become a significant phenomenon. So far, patents have not had a major negative effect on software innovation in general or on open source development in particular. Nevertheless, open source developers are right to be deeply concerned about the possible emergence of patent thickets. Large firms competing with open source developers have little reason not to use patents as a strategic weapon. And a feasible defense strategy for open source developers seems problematic at best.

This problem largely arises because so many patents are granted for trivial inventions that are simply not significant advances over existing knowledge. Low standards allow firms to acquire large patent portfolios with expenditure of money but without commensurate innovation. For an example of a trivial invention, see the patent abstract shown in figure 2-3. This is a patent for a method

of training janitors using illustrated manuals. Apparently, because the manuals have pictures, the patent examiners consider this a nonobvious "invention." They must not have read *Highlights Magazine for Children* in their youth. If they had, they would have found prior art in the "Goofus and Gallant" feature, which shows cartoons of the right and wrong way to do things. Another example is a recent patent for a method of swinging sideways on a backyard swing (No. 6,368,227).[59]

Now in fairness, these are unusually bad patents. But they illustrate the nature of the problem. First, the U.S. Patent and Trademark Office (USPTO) rarely corrects its mistakes. Even though both of these patents have been widely publicized, the patent office has only recently decided to take another look at one of them (the swing patent). Second, if the nonobviousness standard is low enough for backyard swings and illustrated textbooks, it is no obstacle at all for more technically obscure patent applications. Third, even though these patents might very well be found invalid if tested in court, litigation is very expensive. When large firms build portfolios of thousands of patents, it hardly matters whether many of them would survive a court challenge. The small developer cannot afford to find out.

The patent office, in fact, has little incentive to turn down patents since it runs on fees from successful applications. Not surprisingly, the USPTO declares that its quality goal is to "satisfy their customers"; their customers are not, however, society at large.[60] Not surprisingly, very few patents are rejected; if continuations (reapplications) are included, some 95 percent of applications are eventually approved, and few are ever subsequently reexamined.[61]

Some people claim that this problem could be solved if the USPTO had more money. Others suggest that patent quality will improve once the patent office develops better skills in new technology areas.[62] In fact, the USPTO just follows the standards set by the courts in this matter—for it is the courts that dramatically lowered standards. According to the 1952 statute, a patent should not be obvious to "a person having ordinary skill in the

art." But since 1982, when a specialized court (the Court of Appeals for the Federal Circuit) was created to hear all patent appeals, this requirement has been dramatically eased.[63]

Court doctrine now pays great attention to "secondary issues," making it much harder to find a patent invalid because it is obvious. Prior to 1982, in court cases challenging the validity of patents, about 45 percent of the patents were found invalid for obviousness.[64] By the mid-1990s, only 5 percent were found invalid for obviousness. In effect, the federal circuit court has set a policy that permits large numbers of trivial patents in software and in many other fields as well. This policy threatens the future of open source.

Moreover, these government policies cannot be justified as necessary for the promotion of innovation in software. It is often argued that patents are necessary to promote innovation. In the words of the USPTO, "Through the issuance of patents, we encourage technological advancement by providing incentives to invent, invest in, and disclose new technology worldwide."[65] But in software, there is no evidence that patents have increased incentives to invest in research and development. The software industry was highly innovative before patents. Furthermore, Eric Maskin (Institute for Advanced Study at Princeton) and I found that the firms obtaining the large portfolios of software patents actually *decreased* their research and development spending relative to sales.[66] At best, software patents have thus far had a neutral effect on software innovation; at worst, they have had a significant negative impact on future innovation. And there is simply no evidence that patents are needed to promote innovation in software.

Indeed, large portfolios of trivial patents probably deter innovation even among proprietary developers. In the semiconductor industry, where large firms use patent thickets strategically, small firms end up paying a "patent tax" on innovation to the large firms. Not only must small firms pay royalties (IBM collects over $1 billion in royalties each year and Texas Instruments collected royalties over half a billion dollars per year by

the mid-1990s), but the large portfolio holders gain access to the innovator's technology (so-called design freedom).[67] While the negative effect of these burdens has yet to be verified empirically, there seems little doubt that they reduce incentives for emerging firms to innovate. But although small proprietary firms may be taxed, open source developers potentially face extinction because they cannot even afford small patent portfolios and the costs of negotiating or litigating over them.

The prospects for open source and for software innovation generally would be much stronger if the radical policy changes imposed by the courts were reversed. Two sorts of changes would be helpful. First, Congress could restore the subject matter limitations on patents that largely prevented patents on both software and business methods before the creation of the Court of Appeals for the Federal Circuit. However, since the current crop of patents will last up to twenty years, this may not generate the desired benefits for some time.

A second change would reduce the strategic effectiveness of patent thickets by making their acquisition and maintenance more expensive. For example, rigorous standards for nonobviousness could be reimposed, and the doctrine of equivalents could be strictly limited. Also, a "polluter pays" principle could be applied to legal cases where infringement litigation is brought on patents that are found to be obvious (according to a stricter standard than at present).[68] Another way to clear patent thickets is to reduce their value. Penalties for infringement could be reduced, the litigation process could be simplified to reduce costs (for example, by eliminating the use of preliminary injunctions in infringement cases), and the term for software patents could be shortened.[69] Other considerations might level the playing field, allowing open source developers and individuals to obtain and maintain patents at little cost.[70]

These changes may be difficult, since many of them are likely to be opposed by the influential patent bar. However, the health and growth of one of the nation's most dynamic and technologically important industries is at stake.

Conclusion

Open source is an important organizational innovation in the development of software, one that improves the ability of private agents to meet private needs. It corrects imperfections in the market for proprietary software, and it does so without requiring government intervention through subsidies or procurement preferences. Procurement policies should permit government agencies to purchase open source software on the basis of its merits. In short, open source is a private solution that can and should be allowed to flourish without government intervention.

Unfortunately, the government is already heavily involved: relatively recent changes in the patent system threaten to disrupt open source software development. Over the last two decades, the courts have begun to supplant copyright protection, on which the viability of open source depends, with patent protection. The accumulation of large portfolios of software patents by a few large firms threatens to undermine this important social development. The actions of the courts need to be reversed to encourage the growth of open source development and the health of software innovation in general. Unchecked, software developers will lose much of their freedom to modify and enhance open source code, and with that, society will lose an important source of innovation.

net benefits, I believe it is probably not worth doing—in part, because the government or policymaker can probably do other things that have a higher payoff.

12. See the section "Software and Public Goods" in chapter 4.

13. Certainly private companies, especially larger ones with multiple offices, are concerned with standardization. Sharing documents and working jointly is much more difficult in an environment without standard software. Private firms could equally value "openness." For example, a firm might find that customizing open source software, say Linux, provides a certain degree of standardization but also allows internal departments to shape the software to their own needs.

14. If government-funded software is licensed under the GPL, all other programs that incorporate or extend that software must be licensed under GPL. As a result, the government-funded project could not form the basis for a commercial product—it must remain GPL instead. If the government funded proprietary research, it would remain secret (with source code not shared) and therefore would not be available for others to use. However, the owner of the code would have profit incentives to make the code valuable to users, which might lead to licensing of the code to others. If the government-funded software is licensed under a less restrictive open source license, like BSD, then it can be incorporated into commercial products as well as used in other open source projects. A policy requiring BSD-style licensing for government-funded software research could therefore support both the proprietary *and* the commercial software sectors of the market.

Chapter 2
What Good Is Free Software?

1. See the Netcraft Web Server Survey, "Market Share for Top Servers across All Domains, August 1995–July 2002" (www.netcraft.com/survey/ [August 2002]). "Open source" software refers to software in which the source code that programmers use to create the software is freely accessible. This means that the product is freely available to all users and that any programmer can modify, debug, and enhance the product.

2. Audris Mockus, Roy T. Fielding, and James Herbsleb, "A Case Study of Open Source Software Development: The Apache Server," paper prepared for the 22d International Conference on Software Engineering, Limerick, Ireland (http://opensource.mit.edu/papers/mockusapache.pdf [August 2002]).

3. This model is developed more formally in James Bessen, "Open Source Software: Free Provision of Complex Public Goods," ROI Working Paper, July 2002 (www.researchoninnovation.org/opensrc/pdf [August 2002]). This formal analysis is quite similar to the explanation provided by Nikolaus Franke and Eric von Hippel, "Satisfying Heterogeneous User Needs via Innovation Toolkits: The Case of Apache Security Software," Working Paper 4341-02, Sloan School

of Management, Massachusetts Institute of Technology, January 2002 (http://opensource.mit.edu/papers/frankevonhippel.pdf [August 2002]).

4. Josh Lerner and Jean Tirole, "The Simple Economics of Open Source," Working Paper 7600 (Cambridge, Mass.: National Bureau of Economic Research, March 2000); Justin Pappas Johnson, "Economics of Open Source Software," unpublished working paper, May 17, 2001; Jennifer Kuan, "Open Source Software as Consumer Integration into Production," unpublished working paper, October 26, 2000. In addition, researchers from a variety of other disciplines have studied open source software. A collection of working papers, including these, is available at http://opensource.mit.edu/papers [August 2002].

5. Robert Parker and Bruce Grimm, *Recognition of Business and Government Expenditures for Software as Investment: Methodology and Quantitative Impacts, 1959–98,* report presented at the meeting of the Bureau of Economic Analysis, May 5, 2000 (www.bea.doc.gov/bea/papers/software.pdf [August 2002]).

6. For a general history, see Glyn Moody, *Rebel Code: The Inside Story of Linux and the Open Source Revolution* (Cambridge, Mass.: Perseus Publishing, 2001).

7. Richard Stallman, "The Free Software Definition" (www.fsf.org/philosophy/free-sw.html [August 2002]).

8. For a review, see Erik Brynjolfsson and Lorin M. Hitt, "Beyond Computation: Information Technology, Organizational Transformation and Business Performance," *Journal of Economic Perspectives,* vol. 14, no. 4 (2000), pp. 23–48.

9. See, for example, David S. Evans, "Is Free Software the Wave of the Future?" *Milken Institute Review*, 4th quarter (2001), p. 41. His current views, however, appear to have changed.

10. "Out in the Open," *Economist*, April 14, 2001, special section, p. 8.

11. Netcraft Web Server Survey, "Market Share." See also David A. Wheeler, "Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!" 2002 (www.dwheeler.com/oss_fs_why.html [August 2002]).

12. Barton P. Miller and others, "Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services," working paper, University of Wisconsin, 1995 (ftp://grilled.cs.wisc.edu/technical_papers/fuzz-revisited.pdf); Kuan, "Open Source."

13. "The Revenge of the Hackers," *Economist,* July 9, 1998, p. 63.

14. Karim R. Lakhani, Bob Wolf, and Jeff Bates, "The Boston Consulting Group Hacker Survey," January 31, 2002 (www.osdn.com/bcg/bcg/bcghackersurvey.html [August 2002]).

15. Evans, "Free Software," p. 41.

16. See www.kde.org and www.gnome.org.

17. See Rex Baldazo, "CNET Review: Mozilla 1.0 Release Candidate 2," May 14, 2002 (www.cnet.com/software/0-3227883-8-9895059-1.html?tag=st.sw.3227883.bhed.3227883-8-9895059-1 [August 2002]), and Andrew Leonard, "Mozilla's Revenge," *Salon*, March 12, 2002 (www.salon.com/tech/col/leon/2002/03/12/mozilla/ [August 2002]).

18. On the other hand, there is a model that shows that open source has a

comparative advantage over the proprietary provision of complex applications. See Bessen, "Open Source Software." This model suggests that the technically sophisticated nature of much open source software arises from an economic decision, not a fundamental limitation in the open source development process.

19. www.sourceforge.net.

20. Wylie Wong, "Application Server Giants Regroup," *CNET*, December 3, 2001 (www.news.com.com/2100-1001-276459.html?legacy=cnet&tag=st.ne.ni.gartnercomm.ni [August 2002]).

21. Ibid.

22. See also Lerner and Tirole, "Simple Economics of Open Source."

23. For example, David Coursey, "And Today, Microsoft Is Still Driving Me Nuts (Part Two)," May 7, 2001 (www.zdnet.com/anchordesk/stories/story/0,10738,2715734,00.html [August 2002]).

24. These issues are discussed in Michael A. Cusumano and Richard W. Selby, *Microsoft Secrets: How The World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People* (Simon and Schuster, 1995).

25. See Bessen, "Open Source Software"; and Cusumano and Selby, *Microsoft Secrets,* especially p. 310.

26. Generally, asymmetric information describes a situation where one party (or group of parties) has private information that another party (or parties) does not have.

27. There is an additional factor limiting provision under custom proprietary development. In many cases custom development uses an application program interface (API) provided by a standardized software firm. However, monopoly pricing of this API implies a deadweight loss—some consumers will be priced out of the market. See Bessen, "Open Source Software."

28. Franke and von Hippel, "Satisfying Heterogeneous User Needs," provide some evidence for this.

29. Ibid. Security features represent only a fraction of Apache's total feature set, so presumably the total extent of customization is even greater.

30. "Apache Module Registry" (http://modules.apache.org/ [May 25, 2002, with duplicates and bad records eliminated]).

31. "Apache Module Report" (https://secure1.securityspace.com/s_survey/data/man.200204/apachemods.html [May 25, 2002]).

32. Mockus, Fielding, and Herbsleb, "A Case Study."

33. Ibid., table 1.

34. Note that very little of the customization effort can be attributed to firms attempting to economize by using a free product and then correcting deficiencies through customization. The second most popular web server, Microsoft's IIS, is free for users of the Windows operating system. Apache runs on Linux (free), proprietary Unix, and also on Windows. If one assumes that these operating systems are equivalent for running web servers, then Apache offers no direct cost saving relative to IIS. Even if Linux were inferior to Windows but could be fixed through customization of Apache, the cost difference would be

minor—the price of Windows is $300 or less per license. Thus few firms would plausibly customize Apache to compensate for major deficiencies in Linux.

35. "IIS-Enabled Products from Industry Partners" (www.microsoft.com/windows2000/partners/iis.asp [May 25, 2002]).

36. See Johnson, "Economics of Open Source."

37. Of course, some proprietary products experience "network externalities" and may also face similar chicken-and-egg problems.

38. See endnotes 12 through 14.

39. www.sourceforge.net.

40. "Linux Takes on MS in China," *BBC News,* January 8, 2002 (http://news.bbc.co.uk/hi/english/sci/tech/newsid_1749000/1749441.stm [August 2002]).

41. Rick Perera, "Open-source Fans Welcome French Government," *ITWorld.com*, November 26, 2001 (www.itworld.com/Man/2685/IDG011126frenchopensource/ [August 2002]).

42. See http://niap.nist.gov/cc-scheme/historical-perspective.html [August 2002]).

43. Jonathan Krim, "Open-Source Fight Flares at Pentagon: Microsoft Lobbies Hard against Free Software," *Washington Post*, May 23, 2002, p. E1.

44. Ken Brown argues that open source poses a security risk; see *Opening the Open Source Debate* (Washington: Alexis de Tocqueville Institute, June 2002). However, the computer security community has, if anything, a preference for open source for secure systems. See Ross Anderson, "Security in Open versus Closed Systems—The Dance of Boltzmann, Coase and Moore," and Roger Needham, "Security and Open Source," both papers presented at the conference "Open Source Software: Economics, Law and Policy," Institut de Economie Industrielle (IDEI), Toulouse, France, June 20–21, 2002 (www.idei.asso.fr/english/epresent/index.html [August 2002].)

45. The importance of this point has been raised recently by open source advocates at www.sincerechoice.com.

46. Paul Krugman, *Competitiveness: An International Reader* (New York: Foreign Affairs, 1994).

47. SourceForge lists numerous "me too" open source projects that are quite similar to already existing programs. Not surprisingly, few of these receive sustained support.

48. To obtain a patent, an invention is supposed to be nonobvious to a practitioner skilled in the relevant art. See Robert P. Merges, *Patent Law and Policy: Cases and Materials*, 2d ed. (Charlottesville, Va.: The Mitchie Company, 1997); Samuel Kortum and Josh Lerner, "What Is behind the Recent Surge in Patenting?" *Research Policy,* vol. 28, no. 1 (1999), pp. 1–22; Bronwyn H. Hall and Rosemary Ham Ziedonis, "The Patent Paradox Revisited: An Empirical Study of Patenting in the U.S. Semiconductor Industry, 1979–1995," *RAND Journal of Economics,* vol. 32, no. 1 (2001), pp. 101–28; Josh Lerner, "Patenting in the Shadow of Competitors," *Journal of Law and Economics,* vol. 38, no. 2 (1995), pp. 463–95; Glynn S. Lunney Jr., "E-Obviousness," *Michigan Telecommunications and Technology Law Review*, vol. 7 (Fall 2000–Spring 2001), p. 363.

49. "Hearings on Software Patent Protection," U.S. Patent and Trademark Office, January–February 1994.

50. Jean Lanjouw and Mark Schankerman, "Enforcing Intellectual Property Rights," Working Paper 8656 (Cambridge, Mass.: National Bureau of Economic Research, December 2001).

51. Wesley M. Cohen, Richard R. Nelson, and John P. Walsh, "Protecting Their Intellectual Assets: Appropriability Conditions and Why U.S. Manufacturing Firms Patent (or Not)," Working Paper 7552 (Cambridge, Mass.: National Bureau of Economic Research, February 2000).

52. Only 51 percent of corporate patent portfolios were used according to Edwin Mansfield, *The Economics of Technological Change* (W. W. Norton, 1968), p. 207. Rossman and Sanders reported that only 31 percent were in use at the time of their survey, although about half were used at some time; see Joseph Rossman, and Barkev Sanders, "The Patent Utilization Study," *Patent, Trade-Mark, and Copyright Journal of Research and Education*, vol. 1, no. 1 (1957), pp. 74–111. And a 1998 study found that about 60 percent were used; see "Intellectual Property Rights Benchmark Study" (London: Business Planning and Research International, 1998), cited in "The Patent and License Exchange: Enabling a Global IP Marketplace," Harvard Business School Case Study N9-601-019 (July 17, 2000).

53. Peter C. Grindley and David J. Teece, "Managing Intellectual Capital: Licensing and Cross-Licensing in Semiconductors and Electronics," *California Management Review*, vol. 39, no. 2 (1997), pp. 8–41; Hall and Ziedonis, "Patent Paradox Revisited," pp. 101–28.

54. See, for example, Bruce Perens, "Preparing for the Intellectual-Property Offensive," *LinuxWorld* (www.linuxworld.com/linuxworld/lw-1998-11/lw-11-thesource.html [August 2002]); Karston M. Self, "Cooperative OSS Patent Pool—Proposal" (http://home.netcom.com/~kmself.osspatpool/index.html [August 2002]).

55. Quoted in Fred Warshofsky, *The Patent Wars* (Wiley, 1994), p. 170.

56. "IBM Public License Version 1.0" (http://oss.software.ibm.com/developerworks/opensource/license10.html [August 2002]).

57. See www.openpatents.org (August 2002).

58. There have been some notable examples, including a leading developer of free 3D software, Helmut Dersch. See Craig Bicknell, "Virtual Reality, Real Trouble," *Wired News*, July 20, 1999 (www.wired.com/news/ipo/0,1350,20824,00.html [August 2002]).

59. For information on patents, see www.uspto.gov/main/patents.htm (August 2002).

60. This declaration appears on a sign in the lobby of the U.S. Patent and Trademark Office. See also USPTO, "FY2001 Corporate Plan," 2000 (www.uspto.gov/web/offices/com/corpplan [August 2002]), which states, "The Patent Business is one of the PTO's three core businesses. The primary mission of the Patent Business is to help customers get patents." Brian Kahin provides a political

economic analysis of the USPTO in "The Expansion of the Patent System: Politics and Political Economy," *First Monday*, vol. 6, no. 1 (2001).

61. Cecil D. Quillen Jr. and Ogden H. Webster, "Continuing Patent Applications and Performance of the U.S. Patent Office," *Federal Circuit Bar Journal*, vol. 11, no. 1 (2001), pp. 1–21; Stuart Graham and others, "Post-Issue Patent 'Quality Control': A Comparative Study of US Patent Re-examinations and European Patent Oppositions," Working Paper 8807 (Cambridge, Mass.: National Bureau of Economic Research, February 2002).

62. See Adam B. Jaffe, "The U.S. Patent System in Transition: Policy Innovation and the Innovation Process," *Research Policy*, vol. 29, no. 4–5 (2000), pp. 531–57.

63. Lunney, "E-Obviousness."

64. Ibid. These figures are obtained by multiplying Lunney's data for figure 1 (Percentage of Patents Held Invalid Where Validity at Issue and Decided) by the data in figure 2 (Percentage of Invalid Patents Found Invalid for Obviousness).

65. U.S. Patent and Trademark Office, "Our Business: An Introduction to the PTO" (www.uspto.gov/web/menu/intro.html [August 2002]).

66. James Bessen and Eric S. Maskin, "Sequential Innovation, Patents and Imitation," Department of Economics Working Paper 00-01 (Massachusetts Institute of Technology, January 2000).

67. Grindley and Teece, "Managing Intellectual Capital."

68. This idea has been proposed by Jean-Paul Smets, "Stimulating Competition and Innovation in the Information Society" (www.pro-innovation.org/rapport_brevet/brevets_plan-en.pdf [August 2002]). Another approach is to make patent renewals much more expensive so that only truly important innovations would have long patent terms. See Francesca Cornelli and Mark Schankerman, "Patent Renewals and R&D Incentives," *RAND Journal of Economics*, vol. 30, no. 2 (1999), pp. 197–214. This, however, would work against open source developers.

69. Randall Davis and others, "A Manifesto Concerning the Legal Protection of Computer Programs," *Columbia Law Review,* vol. 94 (December 1994), p. 2318. The current term for patents is twenty years, while software is typically amortized over thirty months.

70. Currently, patent fees are lower for individuals. However, something more far-reaching is required to provide equal access since search costs and legal fees are typically expensive.

Chapter 3
Politics and Programming: Government Preferences for Promoting Open Source Software

1. For further details concerning governments that have or are considering policies to promote open source software, see David S. Evans and Bernard Reddy,