Journal of Economic Perspectives—Volume 19, Number 2—Spring 2005—Pages 99–120

The Economics of Technology Sharing: Open Source and Beyond

Josh Lerner and Jean Tirole

The open source process of production and innovation seems very unlike what most economists expect. Private firms usually pay their workers, direct and manage their efforts, and control the output and intellectual property thus created. In an open source project, however, a body of original material is made publicly available for others to use, under certain conditions. In many cases, anyone who makes use of the material must agree to make all enhancements to the original material available under these same conditions. This rule distinguishes open source production from, say, material in the public domain and "shareware." Many of the contributors to open source projects are unpaid. Indeed, contributions are made under licenses that often restrict the ability of contributors to make money on their own contributions. Open source projects are often loosely structured, with contributors free to pursue whatever area they feel most interesting. Despite these unusual features, recent years have seen a rise of major corporate investments into open source projects; for instance, IBM is reported to have spent over $1 billion in 2001 alone on such projects.[1]

The most prominent example of open source production is software, which involves developers at many different locations and organizations sharing code to

[1] See ⟨http://news.com.com/2100-1001-825723.html⟩ (accessed March 21, 2004).

■ *Josh Lerner is Jacob H. Schiff Professor of Investment Banking, Harvard Business School, Boston, Massachusetts, and Research Associate, National Bureau of Economic Research, Cambridge, Massachusetts. Jean Tirole is Research Director, Institut d'Economie Industrielle, University of Social Sciences of Toulouse, Toulouse, France, and Visiting Professor, Massachusetts Institute of Technology, Cambridge, Massachusetts.*

develop and refine computer programs. The importance of open source software can be illustrated by considering a few examples. The market for server software, which is used by the computers that make web pages available to users through the Internet, has been dominated by the open source Apache project since the inception of systematic tracking by Netcraft in 1995. As of March 2004, more than two-thirds of servers employed this or other open source products, rather than commercial alternatives from Microsoft, Sun and other firms. The open source operating system called Linux accounts for 23 percent of the operating systems of all servers; moreover, Linux has rapidly outstripped Microsoft's Windows program as the operating system most frequently embedded into products ranging from mobile phones to video recording devices.[2] Open source software is dominant in a number of other areas as well; for example, PERL and PHP are the dominant scripting languages.

Open source software seems poised for rapid growth in the future. A recent survey of chief information officers suggests that Linux will play an increasingly important role as the operating system for web servers. Linux also has plenty of room to grow in the market for desktop operating systems; at the end of 2003, only 1.4 percent of the queries to Google came from machines running Linux, although that share was rising.[3] The dissemination of open source databases remains in its infancy, but these are projected to become by 2006 significant challengers to commercial systems sold by firms such as IBM and Oracle.[4] As of March 2004, the website SourceForge.net, which provides free services to open source software developers, listed over 78,000 open source projects.

The article reviews the intriguing and rapidly growing phenomenon of open source production. After describing briefly the origins of open source software, we examine the incentives and roles of the various actors in the open source process. We end by highlighting how exploring open source can help us understand other economic problems, as well as considering the prospects of the open source model spreading to other industries and the parallels between open source and academia.

[2] On web server software and Apache, see ⟨http://news.netcraft.com/archives/web_server_survey.html⟩ (accessed March 21, 2004). On the use of Linux in web server operating systems, see ⟨http://www.pcworld.com/news/article/0,aid,112840,00.asp⟩ (accessed March 31, 2004). On the use of Linux for embedded software, see ⟨http://www.linuxdevices.com/articles/AT8693703925.html⟩ (accessed March 21, 2004).

[3] For the survey of chief information officers, see ⟨http://www.morganstanley.com/institutional/techresearch/pdfs/ciosurvey1203.pdf⟩ (accessed March, 21, 2004). On Linux software used for Google searches, see ⟨http://www.internetnews.com/dev-news/article.php/3302941⟩ (accessed March 31, 2004).

[4] The challenge is expected to be led by MySQL, which received a $16 million financing from the venture capital organizations Accel and Benchmark in 2003. MySQL provides its program for free under an open source license and for a substantial fee under a commercial license. See ⟨http://www.informationweek.com/story/showArticle.jhtml?articleID=18312009⟩ (accessed August 8, 2004).

A Brief History of Open Source Software

Software development has a tradition of sharing and cooperation. But in recent years, both the scale and formalization of the activity have expanded dramatically with the widespread diffusion of the Internet. We will highlight three distinct eras of cooperative software development.[5]

During the first era, the 1960s and 1970s, many of the key features of computer operating systems and the Internet were developed in academic settings such as Berkeley and MIT, as well as in central corporate research facilities where researchers had a great deal of autonomy, such as Bell Labs and Xerox's Palo Alto Research Center. Software can be transmitted in either "source code" or "object (or binary) code." Source code is the code using languages such as Basic, C and Java. Object, or binary, code is the sequence of 0s and 1s that directly communicates with the computer, but which is difficult for programmers to interpret or modify. Most commercial software vendors today provide users only with object or binary code; when the source code is made available to other firms by commercial developers, it is typically licensed under very restrictive conditions. However, in this first era, the sharing by programmers in different organizations of the source code for computer operating systems and for widely used transmission protocols was commonplace. These cooperative software development projects were undertaken on a highly informal basis. Typically, no efforts to delineate property rights or to restrict reuse of the software were made. This informality proved to be problematic in the early 1980s, when AT&T began enforcing its (purported) intellectual property rights related to the operating system software UNIX, to which many academics and corporate researchers at other firms had made contributions.

In response to the threats of litigation over UNIX, efforts to formalize the ground rules behind the cooperative software development process emerged, which ushered in the second era. The critical institution during this period was the Free Software Foundation, begun by Richard Stallman of the MIT Artificial Intelligence Laboratory in 1983. The foundation sought to develop and disseminate a wide variety of software without cost. The Free Software Foundation introduced a formal licensing procedure, called a General Public License, for a computer operating system called GNU. (The name GNU is a recursive acronym which stands for "GNU's Not UNIX.") In keeping with the philosophy of the organization that this software should be free to use, free to modify and free to redistribute, the license aimed to preclude the assertion of copyright or patent rights concerning cooperatively developed software. Also, in exchange for being able to modify and distribute the GNU software, software developers had to agree to a) make the source code freely available (or at a nominal cost) to whomever the program is

[5] This history is highly abbreviated. See Lerner and Tirole (2002) and the sources cited therein for a longer account.

distributed; and b) insist that others who use the source code agree to do likewise. Furthermore, all enhancements to the code—and even in many cases code that intermingled the cooperatively developed software with that developed separately—had to be licensed on the same terms. This kind of license is sometimes called "copyleft," because if copyright seeks to keep intellectual property private, copyleft seeks to keep intellectual property free and available. These contractual terms are distinct from "shareware," where the binary files, but not the underlying source code, are made freely available, possibly for a trial period only. The terms are also distinct from public domain software, where no restrictions are placed on subsequent users of the source code: those who add to material in the public domain do not commit to put the new product in the public domain. Some projects, such as the Berkeley Software Distribution (BSD) effort, took alternative approaches during the 1980s. The BSD license also allows anyone to copy freely and modify the source code, but it is much less constraining than the General Public License: anyone can modify the program and redistribute it for a fee without making the source code freely available as long as they acknowledge the original source.

The widespread diffusion of Internet access in the early 1990s led to the third era, which saw a dramatic acceleration of open source activity. The volume of contributions and diversity of contributors expanded sharply, and numerous new open source projects emerged, most notably Linux, an operating system related to UNIX, developed by Linus Torvalds in 1991. Another innovation during this period was the proliferation of alternative approaches to licensing cooperatively developed software. In 1997, a number of individuals involved in cooperative software development adopted the "Open Source Definition." These guidelines took an ecumenical approach to licenses: for instance, they did not require that proprietary code compiled with the open source software become open source software as well.

The Actors' Strategies in Open Source

The key actors in an open source product are the individual contributors and for-profit companies. Both sets of actors respond to the legal incentives embodied in open source production. We will take up the individual contributors, for-profit firms and legal incentives in turn.

What Motivates Open Source Contributors?

The decision to contribute without pay to freely available software may seem mysterious to economists. However, the standard framework of labor economics can be adapted to capture activity in the open source environment (Lerner and Tirole, 2002).

The unpaid programmer working on an open source software development project faces a variety of benefits and costs. The programmer incurs an opportunity cost of time, which can manifest itself in different ways. For example, a programmer

who works as an independent on open source projects forgoes the monetary compensation that could otherwise be earned by working for a commercial firm or a university. For a programmer with a commercial company, university or research lab affiliation, the opportunity cost of working on open source software comes from not focusing on other tasks. For example, the academic's research output may sag and the student's progress toward a degree slow down.

Several short- or long-run benefits may counter these costs. First, open source programmers may improve rather than reduce their performance in paid work. This outcome is particularly relevant for system administrators looking for specific solutions for their company. Second, the programmer may find intrinsic pleasure if choosing a "cool" open source is more fun than a routine task set by an employer. Third, in the long run, open source contributions may lead to future job offers, shares in commercial open source-based companies, or future access to the venture capital market, and last (but not least) ego gratification from peer recognition. Of course, different programmers may put different values on monetary or personal payoffs and on short-term or long-term payoffs.

Economic theory suggests that long-term incentives for working on an open source project are stronger under three conditions: 1) the more visible the performance to the relevant audience (peers, labor market and venture capital community); 2) the higher the impact of effort on performance; and 3) the more informative the performance about talent (for example, Holmström, 1999).[6] The first condition gives rise to what economists call "strategic complementarities." To have an "audience," programmers will want to work on software projects that will attract a large number of other programmers. This argument suggests the possibility of multiple equilibria. The same project may attract few programmers because programmers expect that other programmers will not be interested; or it may flourish as programmers (rationally) have faith in the project.

To compare programmers' incentives in the open source and proprietary settings, we need to examine how the features of the two environments shape incentives. From the standpoint of the individual, commercial projects typically offer better current compensation than open source projects, because employers are willing to offer salaries to software programmers in the expectation that they will capture a return from a proprietary project. Yet even commercial firms that compensate programmers may want their employees to work on open source projects. Besides the strategic reasons described below, we already noted that the impossibility of appropriating one's contribution to an open source project can be offset if the activity brings private benefits like the ability to fix bugs and customize the product to one's own ends for the programmer. (Commercial software vendors—like Microsoft in its shared source initiative—have sometimes tried to

[6] For a discussion as to how firms might otherwise have superior information about employees and how this might deter job offers from outsiders—a problem that open source programming can address—see Greenwald (1986) and Waldman (1984).

emulate this benefit by opening their code to selected users under a confidentiality arrangement.) Also, open source code may already be familiar to programmers: because it is freely available to all, it can be used in schools and universities for learning purposes, thus creating an "alumni effect." (Again, commercial software vendors are trying to emulate this benefit through university licenses to, say, Windows code.)

When we consider the delayed rewards of working on an open source project, the ability to signal a high level of competence may be stronger in the open source mode for three reasons. First, in an open source project, outsiders can see the contribution of each individual, whether that component "worked," whether the task was hard, if the problem was addressed in a clever way, whether the code can be useful for other programming tasks in the future, and so forth. Second, the open source programmer takes full responsibility for the success of a subproject, with little interference from a superior, which generates information about ability to follow through with a task. Finally, since many elements of the source code are shared across open source projects, more of the knowledge they have accumulated can be transferred to new environments, which makes programmers more valuable to future employers.

These incentives are likely to be stronger and the project more successful if there is an effective leader. While the leader of an open source project has no formal authority—that is, he cannot direct any one to do anything—the leadership often has considerable "real authority," in the terminology of Aghion and Tirole (1997). Leaders play a key role in formulating the initial agenda, setting goals as the project evolves and resolving disputes that might lead to the splintering or outright cessation of the project.

The empirical evidence is largely consistent with the belief that individual contributors to open source projects do benefit directly. The sole econometric study we are aware of, by Hann, Roberts, Slaughter and Fielding (2004), examines contributors to the Apache project, drawing on a wide variety of project records. The authors complement these data with a survey on employment, which yield useable data for multiple years for 147 contributors to the project. The authors then estimate a series of regressions, in which they use the logarithm of earnings in a given year as the dependent variable, and information on the respondents' background, work experience and contributions to and current position in the Apache project in the previous year as independent variables. While in a number of projects such as Linux there is an undisputed central leader, in Apache and many other projects a series of committees at different levels resolves open issues. As a result, there are five observable levels of recognition or rank within the Apache Software Foundation (ASF), which runs the project: in order of increasing status, the titles are developer, committer, project management committee member, ASF member and ASF board member. Advancement is made in recognition of an individual's commitment and contributions to an Apache project. To control for unobserved individual characteristics, individual fixed effects are employed. The

results suggest that the sheer volume of contributions to the Apache project have little impact on salary. But individuals who attain high rank in the Apache organization enjoy wages that are 14 to 29 percent higher, whether or not their work directly involves the Apache program. The results appear to be robust to controls for the possible alternative explanations: for instance, the authors address the possibility that Apache promotions may be driven by commercial success by using lags of the key independent variables.

Academics have also attempted to understand motivations of those who work on open source projects through surveys. Given the inherent subjectivity of these assessments and the self-serving biases in reporting, the low response rates that many of these surveys have obtained, and the sensitivity of some of these questions, it is perhaps not surprising that self-reported motivations vary considerably across studies. For instance, Haruvy, Wu and Chakravarty (2003) find that commercial objectives—particularly, the promise of higher future earnings—are an important driver of contributions to open source projects. However, Lakhani and von Hippel (2003) suggest that the overwhelming driver of open source contributors is the need to solve their own specific programming needs, while a Boston Consulting Group (2003) survey implies that intellectual curiosity is the most important determinant.

How Do Commercial Firms Work and Compete with Open Source?

Commercial companies may interact with an open source project in a number of ways. While improvements in the open source software are not appropriable, commercial companies can benefit if they also offer expertise in some proprietary segment of the market that is complementary to the open source program. Also, firms may temporarily encourage their programmers to participate in open source projects to learn about the strengths and weaknesses of this development approach. For-profit firms may compete directly with open source providers in the same market. Finally, commercial companies may interface with the open source world because it generates good public relations with programmers and customers.

A for-profit firm that seeks to provide services and products that are complementary to the open source product, but are not supplied efficiently by the open source community, can be referred to as "living symbiotically." IBM, which has made open source software into a major focus for its consulting business, exemplifies this approach. A commercial company in this situation will want to have extensive knowledge about the open source movement, and may even want to encourage and subsidize open source contributions, both of which may cause it to allocate some programmers to the open source project. Because firms do not capture all the benefits of the investments in the open source project, however, the free-rider problem often discussed in the economics of innovation should apply here as well. Subsidies by commercial companies for open source projects should remain somewhat limited.

The *code release* strategy arises when companies release some existing proprietary

code and then create a governance structure for the resulting open source development process. For example, IBM released half-a-million lines of its Cloudscape program, a simple database that resides inside a software application instead of as a full-fledged database program, to the Apache Software Foundation. Hewlett-Packard released its Spectrum Object Model-Linker to the open source community to help the Linux community write software to connect Linux with Hewlett Packard's RISC computer architecture. This strategy is to give away the razor (the released code) to sell more razor blades (the related consulting services that IBM and Hewlett Packard hope to provide).[7]

When can it be advantageous for a commercial company to release proprietary code under an open source license? In general, it will make sense if the increase in profit in the proprietary complementary segment offsets any profit that would have been made in the primary segment, had it not been converted to open source. Thus, the temptation to go open source is particularly strong when the product is lagging behind the leader and making few profits, but the firm sees a possibility that if the released code becomes the center of an open source project and is utilized more widely, the profitability of the complementary segment will increase. (An example may be Netscape's 1998 decision to make "Mozilla," a portion of its browser source code, freely available.) If network effects and switching costs are very strong, the second-best commercial package might have a tiny market share. In these cases, the cost to corporations of releasing code may be very small. Moreover, such a strategy may reassure potential users that the released software will never be withdrawn—and thus that the user will always be able to maintain the product itself.

This motivation can also depend on the evolution of vertical relationships between small and large firms in the software industry in commercial software environments, a subject that would reward further study. Indeed, many small developers are uncomfortable doing business with leading software firms. They fear that the commercial platform owner has an incentive to introduce substitutes in the developers' segment to force prices down in that segment and to raise the demand for licenses to the broad software platform (Farrell and Katz, 2000). By contrast, when a large firm makes its platform available on an open source basis through (say) a General Public License-style license, the small firm need no longer fear being squeezed in this way.

Numerous challenges appear, though, when a for-profit firm seeks to become the center of an open source development project. Leadership by a commercial entity may not internalize enough of the objectives of the open source community. In particular, a corporation may not be able to make a credible commitment to keeping all source code in the public domain and to highlighting important contributions adequately. These difficulties help to explain why Hewlett-Packard

[7] For more details, see ⟨http://www.infoworld.com/article/04/08/03/HNclouscape_1.html⟩ (accessed August 3, 2004), ⟨http://www.collab.net/customers/cdp_solutions_at_work.html⟩ (accessed March 31, 2004) and the associated links.

released its code through Collab.Net, a venture by leading open source programmers, which organizes open source projects for corporations who wish to open up part of their software. In effect, Collab.Net offers a kind of certification that the firm is committed to the open source project. (The Apache Software Foundation plays a similar role in Cloudscape case mentioned above.) In a theoretical model, Dessein (2002) shows that a principal with formal control rights over an agent's activity in general gains by delegating control rights to an intermediary with preferences or incentives that are intermediate between the principal's and the agent's. The partial alignment of the intermediary's preferences with the agent's fosters trust and boosts the agent's initiative, ultimately offsetting the partial loss of control for the principal. In the case of Collab.Net, the congruence with the open source developers is obtained through the employment of visible open source developers and the involvement of O'Reilly, a technical book publisher with strong ties to the open source community.

While the relative merits of open source and proprietary software are discussed in a number of contributions, direct competition between the two paradigms has received little attention. An exception is Gaudeul (2004), who builds a duopoly model with one open source and one proprietary software project.[8] In his model, open source software has both costs and benefits relative to proprietary software. Open source software suffers from some lack of coordination: the same code may be written twice or not at all. Another cost of open source software in Gaudeul's model is that its designers, the developers, may not bother developing interfaces that appeal to unsophisticated users. By contrast, the profit-maximizing proprietary software firm in his model is keener to develop such an interface. However, the proprietary model must pay its developers and, despite good project coordination, may choose to develop a limited set of features. In this model, the proprietary software is sold to users at a positive price that excludes some possible users. In equilibrium, the open source software, if it survives, is used either by low-demand or low-income consumers, who cannot afford buying the proprietary software, or by developers who like the potentially larger set of features and do not care about the missing or insufficient user interface. Furthermore, the presence of open source software raises welfare, at least if it does not discourage the development of proprietary software with a good interface.

How Does the Legal System Affect Open Source?

Open source software is shaped by the legal rules under which it operates. In each case, the product originator gives users the right to employ the copyrighted code through a license. But the licenses differ tremendously in the extent to which they enable licensors and contributors to profit from the code that is contributed.

In Lerner and Tirole (2005), we explore what drives firms to choose particular

[8] See also the discussion below of Casadesus-Masanell and Ghemawat (2003).

licenses. We begin with a model of license choice. We suppose that an entity, either an individual or a firm, is deciding a) whether to make some software available under an open source license; and b) if so, what type of license to employ. We depict the interactions between the licensor and the community of programmers. The programmers' benefits from working on the project may depend on the choice of license. The licensor must assess how its choice of license, together with project characteristics—such as the environment, the nature of the project and the intended audience—impacts the project's likely success.

The model suggests that permissive licenses such as the Berkeley Software Distribution model, where users retain the ability to use the code as they see fit, will be more common in cases where projects have strong appeal to the community of open source contributors—for instance, when contributors stand to benefit considerably from signaling incentives or when the licensors are well-trusted. Conversely, restrictive licenses such as the General Public License will be commonplace when such appeals are more fragile. Examples of cases where we would expect a restrictive license are projects geared for end users who are unlikely to appreciate the coding, such as computer games, or those sponsored by corporations, who potential contributors might fear would "hijack" the project and use the code for commercial ends.

One of the most visible of the disputes over licensing was the Mozilla case alluded to above. This effort initially encountered severe difficulties because of the license choice. Netscape initially proposed the "Netscape Public License," which would have allowed Netscape to take pieces of the open source code and turn them back into a proprietary project again (Hamerly, Paquin and Walton, 1999). Ultimately, the firm announced the "Mozilla Public License," under which Netscape cannot regain proprietary rights to modifications of the code: in fact, the terms of the final license are even stricter than those of the General Public License.

In Lerner and Tirole (2005), we also present an empirical analysis of the prevalence of different types of open source licenses. The analysis employs nearly 40,000 open source projects in the SourceForge database. Since all of the projects in this database are open source, we focus on whether the license requires that when modified versions of the program are distributed, the source code must be made generally available and/or whether the license restricts modified versions of the program from mingling their source code with other software that does not employ such a license. We term such licenses "restrictive." We find that restrictive licenses are more common for applications geared toward end-users and system administrators—like desktop tools and games. However, restrictive licenses are significantly less common for those applications aimed toward software developers. Restrictive licenses are also less common for projects operating in commercial environments or that run on proprietary operating systems. Projects whose natural language is not English, whose community appeal may be presumed to be much smaller, are more likely to employ restrictive licenses. Projects with less restrictive licenses tend to attract more contributors.

Further Issues about Open Source

This section will highlight three other particularly interesting and challenging areas about open source projects: the quality of their output; whether open source projects should be encouraged by public policy; and how open source projects may be affected by software patents.

What is the Relative Quality of Open Source Software?

One of the most contentious issues in the literature has been the relative virtues of the open source and proprietary development process. Advocates of open source software have long claimed that the open source development process leads to superior software (for example, Raymond, 1999). A number of studies have sought to explore these claims, but consensus remains elusive.

Kuan (2001) was the first to offer a formal model of some of the advantages of open source software for users. She focused on the consumer's choice between employing "off-the-shelf" commercial software and adapting open source code to the consumer's own use. While the proprietary software can (and indeed must) be used "as is," open source code can be enhanced in quality through the user's efforts. Such refinements, however, require effort, with more effort leading to higher quality code. If consumers differ in type, commercial companies may either offer different quality levels of programs, or else may offer a single product to all users. She shows that under certain circumstances, some consumers will prefer the open source option and invest in producing software that is of superior quality to commercial alternatives. The paper tests this model by comparing dates at which program errors or "bugs" were reported and fixed in three open source programs—Apache, FreeBSD and Gnome—with three commercial projects matched by subject matter and age. For two of the three pairs that she examines, the rate at which bugs are fixed is significantly faster in the open source project, and there is little difference in the third case.

Bessen (2002), in a related paper, emphasizes another dimension along which open source software may have an advantage: the ability of heterogeneous users to customize it to meet their own particular needs. Proprietary software manufacturers cannot anticipate all manifestations of consumer demand, and thus cannot offer every conceivable variation that consumers might desire. Again, consumers face a "make versus buy" choice, where the complexity and idiosyncrasy of the project, as well as the cost of modifications, will drive the choice. While Bessen does not test this model, he cites Franke and von Hippel's (2003) finding that one-fifth of Apache users adapted security features to meet their particular needs as consistent with his model.

While these two authors attribute the superiority of open source projects to the ability of end-users to adapt an initial code base, Johnson (2004) focuses on a different rationale: that open source programs are developed through a superior process that may avoid pathologies that affect commercial projects. In particular,

he argues that workers in commercial firms may collude not to report programming errors of fellow employees, lest their own reputation and future earnings be damaged. He hypothesizes that because programmers do not receive wages in open source projects, they will have fewer incentives to engage in such collusion. (Note, though, that the ego gratification and career incentives may also motivate collusion.) It may be argued that the large number of potential eyeballs in open source software makes collusion difficult to sustain. Johnson argues that reduced collusion will lead to open source projects undergoing more peer review and having higher quality.[9]

Another issue that may differentiate proprietary and open source software is security. Open source advocates have argued that when source code is open and freely visible, programmers can readily identify security flaws and other problems: as Eric Raymond (1999) has argued, "to many eyes, all bugs are shallow." Proponents of proprietary software, on the other hand, argue that the openness of the source code allows malicious hackers to figure out its weaknesses. Anderson (2002) argues that under certain plausible assumptions, the openness of the system should have no impact on its security. Making bugs harder for hackers to find by keeping the source code hidden will also mean that software companies have a more difficult time identifying errors through "beta" testing, where lead users experiment with the product, also without access to the underlying source code. (While software firms will also do internal testing by employees with access to the source code, the effort devoted to these "alpha" tests is usually many times smaller than that in later-stage tests.) Thus, he concludes, "other things being equal, we expect that open and closed systems will exhibit similar growth in reliability and in security assurance." However, Anderson does not attempt to assess this claim empirically. Any such effort is difficult because hackers may attack a software program for reasons unrelated to the intrinsic security of the program; for instance, some hackers may derive more gratification from an attack on a leading public company, even though hackers have targeted both commercial and open source programs at various occasions.[10]

Open source and commercial software could be compared as well along numerous other dimensions. For instance, we argue in our 2002 work that it is likely that the incentive structure for open source programmers will lead to poorer documentation and user interfaces in these projects. These claims, and numerous others in the literature, deserve careful scrutiny.

[9] Also in this paper, Johnson (2004) suggests that individuals in commercial software companies may be reluctant to report programming problems to superiors, because the firm's management may be unable to commit not to demand that they then address these issues. In open source projects, programmers can never be compelled to work on fixing a bug that they identify. He predicts that while the speed of the bug fixing process may be slower in open source projects, more problems will ultimately be identified.

[10] For a discussion of a hacker attack on Apache, see ⟨http://thewhir.com/marketwatch/hac062102.cfm⟩ (accessed March 31, 2004).

What are Appropriate Public Policies Toward Open Source?

Government commissions and agencies have proposed—and in some cases implemented—a variety of measures to encourage open source developers. For example, in the United States, the President's Information Technology Advisory Committee (2000) recommended direct federal subsidies for open source projects to advance high-end computing, and a report from the European Commission (2001) also discussed support for open developers and standards. Many European governments have policies to encourage the use and purchase of open source software for government use ("Microsoft at the Power Point," 2003). Governments may even mandate the development of localized open source projects, as has occurred in China (Open Source Development Labs, 2004).

Economists have sought to understand the consequences of a vibrant open source sector for social welfare. Perhaps not surprisingly, definitive or sweeping answers have been difficult to come by; instead, the policy conclusions focus on specific instruments in specific contexts. We will first discuss two papers that consider the impact of open source on social welfare more generally and then discuss a number of works that address public policies.

Most analyses have suggested that government support for open source projects is likely to have an ambiguous effect on social welfare. For example, Johnson (2002) presents a model where programmers decide whether to devote effort to a project, in which their contributions become a public good once they are developed. Users thus face a decision whether to enhance an existing open source program or to wait in the hope that another programmer will undertake the development process. Johnson then compares this process to a stylized depiction of the development of proprietary software in a corporate setting. Open source projects have the advantage of being able to access the entire pool of developer talent, not just employees in a single firm. Given the larger talent pool, they can aggregate and exploit more private information. But because of the free riding problem, some potentially valuable projects will not be developed under an open source system. Johnson concludes that a comparison of the social welfare consequences of these two systems is ambiguous.

Casadesus-Masanell and Ghemawat (2003) depict competition between an open source operating system available at no cost and a proprietary commercial product. The crucial feature of their model is on the demand side: the larger the market share of a given operating system, the more valuable that system to users. This effect could be due to better learning about the program's features (if users contribute comments and suggestions to improve the product) or to the presence of complementary software developed by other firms. In this setting, the presence of an open source operating system leads the commercial firm to set lower prices, which in turn means that the overall use of operating systems is higher. However, the value of the commercial system for users is lower: for instance, the presence of a competing product may lead third-party developers to develop fewer complementary products for the commercial operating system. Thus, the presence of open source projects may either make society

better or worse off. This model also suggests that in some cases, the proprietary operating system may be able to drive the market share of the open source alternative to zero, and also that the parameter ranges where this will occur need not correspond to those where such an action is socially desirable.

Schmidt and Schnitzer (2002) highlight similarly that open source software has social costs and benefits. Building on a line of economic reasoning that extends back to Arrow (1962) and even earlier, they highlight two countervailing effects. From a static point of view, free or nearly free open source will insure greater social welfare, as virtually any potential user will be able to access software. But from a dynamic perspective, with so few profits to be gleaned, developers may lack incentives to introduce new products. While career concerns and other incentives may motivate developers to identify bugs in open source programs and undertake certain modest adaptations to meet their own needs, they are unlikely to be sufficient to encourage major breakthroughs. The authors argue that while open source programs will enhance social welfare in some settings, this outcome will be far from universal. They caution against subsidies that may lead to an undesirably high level of open source activity.

Saint-Paul (2003) reaches an even bleaker conclusion about the open source phenomenon. He employs a Romer-style endogenous growth model, in which both commercial firms and "philanthropists"—individuals who are willing to give their contributions away for free—innovate. He shows that the free contributions will lead to economic growth, but also reduce the profits, and hence the incentives to innovate, among commercial firms. Unless the proprietary sector is quite profitable, then the second effect will dominate, and innovation and growth be harmed by the presence of open source software. He argues that the negative effect is likely to be even stronger than his model shows, because he neglects, for instance, the possibility that philanthropic products do not meet users' needs as well as commercial products (though see the previous section for a counter-argument) and can also divert programming talent that could have been devoted to commercial products.

In a more informal piece, Shapiro and Varian (2004) suggest another consideration that formal models have not so far discussed: the impact on human capital and entrepreneurship. They suggest that an open system will facilitate learning by students as to how to program and will provide opportunities for third-party developers to introduce complementary products. They argue that all else being equal, these considerations should lead public policymakers in nations that seek to encourage the development of their software industries to boost the development of open source activity.

How Will Software Patents Affect Open Source?

Software patents will interact with open source activity.[11] This issue is clearly a timely one, in light of the litigation launched by the SCO Group, which holds to (at

[11] In this section, we will avoid discussing the highly contentious and unsettled question of the economic impact of software patents more generally: for more on this topic, see, for instance, Bessen and Hunt (2003), Caillaud (2003), Graham and Mowery (2003) and Hahn and Wallsten (2003).

least partial) rights to UNIX (acquired from Novell, who in turn had purchased them from AT&T). Beginning in 2003, the firm initiated a series of lawsuits against, among others, AutoZone, DaimlerChrysler, IBM and Novell, alleging that they were violating SCO's intellectual property by contributing to or using Linux.[12] The allegedly detrimental impact of software patents on open source was also a frequently invoked reason for opposing software patents in the September 2003 debate in the European Parliament on this question.[13]

Software patents create the possibility of holding up software producers. In the case of commercial software, individuals and companies that do not produce software themselves (like hardware manufacturers and software users), but hold a software patent, can try to obtain royalty payments from major software vendors. Large software vendors are less likely to engage in such behaviors against each other, as they have accumulated patent portfolios that they can use for retaliatory purposes against other vendors that would try to hold them up.

Open source software is vulnerable in a different way. After all, the code is free of charge and the contributors hardly solvent for the most part, so attempting to collect royalties is not a powerful incentive. However, firms with software patents may seek damages from large corporate and noncorporate users and firms that service open source software, as SCO is doing, or commercial competitors with software patents may sue open source software to enjoin further utilization of the code. It remains to be seen whether the open source movement will itself enter into defensive patenting, as large commercial vendors already do, or at least make a more concerted effort to forestall patenting by others by aggressively publishing. One intriguing new initiative is the Red Hat Assurance Plan, in which the Linux distributor is offering partial protection against intellectual property litigation.[14]

Another interesting area of study concerns the consequences of users paying royalties for an open source program that also included some commercially patented material. Such royalty demands might trigger "sweetheart" deals between firms, the splitting of open source projects into different branches (often termed "forking") and the privatization of blocks of code. The General Public License seeks to address these problems—and to discourage patent filings by firms working with open source projects—by prohibiting the incorporation of code that is encumbered by patent rights. As section 7 of the license states, "[I]f a patent license would

[12] Patent concerns have also slowed the adoption of Linux in the public sector: see, for instance, ⟨http://www.informationweek.com/story/showArticle.jhtml?articleID=26806464⟩ (accessed August 25, 2004) for a discussion of the impact of these concerns on the city of Munich's open source effort.

[13] See, for instance, ⟨http://news.zdnet.co.uk/business/legal/0,39020651,39116053,00.htm⟩ (accessed March 26, 2004).

[14] For a discussion, see ⟨http://www.redhat.com/about/presscenter/2004/press_blackduck.html⟩ (accessed August 24, 2004). One challenge is that the extent and dispersion of the patent holdings that may impact open source projects: the insurer Open Source Risk Management estimates that there are 283 patents that might be used in claims against the Linux kernel alone. See ⟨http://www.eweek.com/article2/0,1759,1631336,00.asp⟩ (accessed August 24, 2004).

not permit royalty-free redistribution 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." Many other types of open source licenses, however, do not address this issue.[15]

Another, less prominent, question relates to the impact of patents on the dynamics of information sharing and collaboration among open source contributors. To what extent will the ability of programmers to protect their discoveries with strong patent rights reduce their incentives to participate in open source projects?

To date, very little systematic analysis has examined the implications of patents for open source. However, a broader literature has scrutinized the impact of patenting on the generation and diffusion of scientific knowledge more generally. Since 1980, a series of reforms in the United States and elsewhere have greatly augmented the ability of academic institutions, government laboratories and nonprofit institutions to patent their discoveries, even if governments originally funded the research. This literature has sought to understand the pervasiveness and consequences of the "anticommons" problem (Heller and Eisenberg, 1998): the concern that the patenting of scientific knowledge will lead to lower research productivity, and hence eventually to reduced economic growth. Much of the discussion of these questions to date has featured broad assertions and anecdotal examples (as in Bok, 2003). It is clear from these studies that institutions and researchers have responded to the increased incentives to commercialize products by engaging in more patenting and commercialization activities (for instance, Jaffe and Lerner, 2001; Lach and Schankerman, 2003). Whether these commercial activities have detrimental effects on research and social welfare is much more ambiguous.[16] Given this initial and somewhat contradictory evidence, our ability to draw conclusions for consequences of formal intellectual property rights for open source software is quite limited.

[15] A related danger is that programs will inadvertently infringe patents. Programmers may lack the incentives and skills needed to check whether their contribution infringes awards. As an effort to limit this problem, beginning in May 2004, Linux contributors were required to attest that they have the right to make that contribution. For a discussion, see ⟨http://www.computerworld.com/softwaretopics/os/linux/story/0,10801,93395,00.html⟩ (accessed August 8, 2004).

[16] For instance, Thursby and Thursby's (2003) study of six major research universities suggests that while the probability that a faculty member will indicate to his university's technology transfer office that he has made a new discovery has increased ten-fold over the past decade, research productivity in basic research journals has remained constant. On the other hand, Murray and Stern (2003) have shown that papers published in the journal *Nature Biotechnology* are somewhat less likely to be cited in other articles once the corresponding patent application issues. They find that the papers with corresponding patents are initially more heavily cited than those without, but then their citation rate declines more sharply over time.

A Broader Research Agenda

The open source process poses numerous interesting issues that extend well beyond the software industry. In this section, we'll highlight three of these. The first issue is the extent to which the open source model can move beyond software into other industries. Second, we'll discuss the way in which firms can accomplish many of the key goals of open source while employing other arrangements. Finally, we'll explore the parallels between open source software and academic research.

Can Open Source Work Beyond Software?

An interesting question is whether the open source model can be transposed to other industries. Could automobile components be developed in an open source mode, with General Motors and Toyota performing an assembler function similar to that of Red Hat for Linux? Many industries involve forms of cooperation between commercial entities in the form of for-profit or not-for-profit joint ventures. Others exhibit user-driven innovation or open science cultures. Thus, a number of ingredients of open source software are not specific to the software industry. Yet no other industry has yet produced anything quite like open source development.

Although some aspects of open source software collaboration (such as electronic information exchange across the world) could easily be duplicated, other aspects would be harder to emulate. Consider, for example, the case of biotechnology. It may be impossible to break up large projects into small manageable and independent modules, and there may not sufficient sophisticated users who can customize the molecules to their own needs. The tasks that are involved in making the product available to the end user involve larger expenditures than simply providing consumer support and friendlier user interfaces as in software. The costs of designing, testing and seeking regulatory approval for a new drug are enormous.

More generally, in many industries, the development of individual components requires large-scale teamwork and substantial capital costs, as opposed to (for some software programs) individual contributions and no capital investment (besides the computer the programmer already has). Another obstacle is that in mass-market industries, users are numerous and rather unsophisticated, and so deliver little peer recognition and ego gratification. These characteristics suggests that the open source model may not easily be transposed to other industries, but further investigation is warranted.

Can Firms Realize the Benefits of Open Source in Other Ways?

Corporations may emulate some of the benefits attached to open source production either by getting involved in open source themselves or by adopting institutional arrangements that deliver some of these benefits. First, using open source technology encourages users that they will not be "held up" by a future price increase after adopting a technology and that they will always be able to tailor their technology to their own particular needs. Second, open source avoids the problem

of a "patent thicket" when multiple firms have overlapping intellectual property rights and at least one party attempts to extract a high fee for its particular contribution. Third, a firm might make a technology open source as a way of trying to certify a technological standard, in which case firms may contribute software to open source to benefit from the endorsement of such a standard, as the Hewlett Packard case discussed above illustrates.

Firms can also address these problems in non-open-source ways, such as patent pools, standard-setting organizations and self-imposed commitments. In a patent pool, firms blend their patents with those of other firms. These pools allow users to access a number of firms' patents simultaneously, thereby avoiding the "patent thicket." In many cases, the pooling agreements also specify the pricing schedule in the agreement that establishes the pool, assuring that no party attempts to extract very high fees or to increase its fees after users are locked in. To be certain, patent pools raise a risk that they can be used to hinder entry, but these concerns can in part be addressed through a careful design of the pool (Lerner and Tirole, 2004a; Lerner, Strojwas and Tirole, 2003).

Standard-setting organizations offer an alternative path for the certification of new technologies. Often firms can choose between standard-setting organizations, and they can seek an endorsement for an emerging technology from an independent and prestigious organization or use a more complacent one (Lerner and Tirole, 2004b). These bodies also help address the other concerns, frequently asking contributors of the key technologies to commit to license the technology on "reasonable and non-discriminatory" terms or to make various other concessions.

Self-imposed commitments can serve much the same role. For instance, firms can commit to license technologies at a given price schedule, or they can commit to provide sufficient information so that users can tailor the technology. An example of the latter is Microsoft's Shared Source Initiative, through the firm shares source code with customers, partners and governments. One open question about many of these self-imposed programs is the extent to which the commitments can be enforced if the firm subsequently changes its design.[17]

Open source production may seem like a unique and idiosyncratic realm. However, many of the issues are seen elsewhere in high-technology industries: when and how to share technology, how to set common standards, and how to combine freely available and commercial components arise both in the open source and the commercial realm. Open source projects and traditional firms can borrow from each other innovative approaches to the underlying problems.

Open Source and Academia

Open source and academia have many parallels. The most obvious parallel relates to motivation. As in open source, the direct financial returns from writing

[17] This is also a question for other commitments as well. For one illustration in a standard setting context, see *Rambus Inc. v. Infineon Techs. A*G, 318 F.3d 1081 (Fed. Cir. 2003).

academic articles are typically nonexistent, but career concerns and the desire for peer recognition provide powerful inducements.

Other similar dynamics are also at work. Consider, for instance, the discussion of motivation for programmers when choosing to which open source project they should contribute. As we highlight above, a critical goal is the selection of a project that is likely to continue to be successful, so that the programmers' contributions are widely recognized, yet which at the same time has interesting and challenging programming challenges to be addressed. These criteria should be familiar to anyone who has advised a doctoral student on the choice of a thesis topic!

However, there are also some substantial differences between the two realms. Here, we will highlight two areas where academic economists could learn from the open source realm. The first of these relates to the incentives to create public goods. Open source contributors often create substantial bodies of code, which are made widely available when completed. Far too often in academic economics, however, we do not see similar dissemination.[18] For instance, an author—after creating a unique dataset for a project—may simply save this information on a personal computer hard disk, rather than making it publicly available. Similarly, while there are some examples of efforts to create shared resources that can be widely used by the economics community—the NBER Patent Citations Database created by Bronwyn Hall, Adam Jaffe and Manuel Trajtenberg is a recent important example—far too often these efforts are neglected because the returns to the project leaders are low. Why it is not commonplace to see economists frequently seeking to establish their reputation by creating original, widely accessible datasets is an interesting question. (Akin to open source, we might anticipate that this strategy might be especially effective for those at smaller and less centrally located institutions.) One explanation might be that data collection is often inspired by what analyses one wants to perform, so it is harder to separate data collection and analysis. In any case, the design of mechanisms that successfully encourage such investments is an important challenge for academic economists.

A second area relates to access to published work. Contributors to open source projects seem to be powerfully spurred by the provisions of these licenses. The assurance that contributions—and subsequent contributions that build on it—will remain publicly accessible incentivizes programmers to write code. By way of contrast, in academic economics, it is standard to assign the copyright to one's work to a commercial publisher. In other areas of academia, this approach is under increasing attack. For instance, recent years have seen the rise of "open access" journals such as the *Public Library of Science*, which make all articles freely accessible and distributable. In response to this challenge, a number of established science journals, such as the *Proceedings of the National Academies of Sciences*, have not only begun providing free access to older issues, but even allowing authors to opt to have

[18] Data archiving policies, such as the *American Economic Review*'s, seek to address this problem, but are more the exception than the rule.

their articles immediately publicly accessible with the payment of an additional fee.[19] It is an interesting question as to whether open access will have the same appeal for the economics community.

Final Thoughts

This paper has reviewed our understanding of the growing open source movement. We have highlighted how many aspects of open source software appear initially puzzling to an economist. Our ability to answer confidently many of the issues raised here questions is likely to increase as the open source movement itself grows and evolves.

At the same time, it is heartening to us how much of open source activities can be understood within existing economic frameworks, despite the presence of claims to the contrary. The labor and industrial organization literatures provide lenses through which the structure of open source projects, the role of contributors and the movement's ongoing evolution can be viewed.

■ *We thank the National Science Foundation and Harvard Business School's Division of Research for financial support. The Institut D'Economie Industrielle receives research grants from a number of corporate sponsors, including France Telecom and the Microsoft Corporation. We thank Christophe Bisiere, Jacques Cremer, Alexandre Gaudeul, Justin Johnson, Hal Varian and the editors for helpful comments.*

[19] See, for instance, ⟨http://www.plos.org/about/openaccess.html⟩ and ⟨http://www.pnas.org/cgi/content/full/101/23/8509⟩ (accessed August 10, 2004).

References

Aghion, Philippe and Jean Tirole. 1997. "Formal and Real Authority in Organizations." *Journal of Political Economy.* 105:1, pp. 1–29.

Anderson, Ross. 2002. "Security in Open versus Closed Systems—The Dance of Boltzman, Coase and Moore." Unpublished working paper, Cambridge University.

Arrow, Kenneth J. 1962. "Economic Welfare and the Allocation of Resources for Invention," in *The Rate and Direction of Inventive Activity: Economic and Social Factors.* Richard R. Nelson, ed. Princeton, N.J.: Princeton University Press, pp. 609–26.

Bessen, James. 2002. "Open Source Software: Free Provision of Complex Public Goods." Unpublished working paper, Research on Innovation.

Bessen, James and Robert M. Hunt. 2003. "An Empirical Look at Software Patents." Working Paper 03-17, Federal Reserve Bank of Philadelphia.

Bok, Derek. 2003. *Universities in the Marketplace: The Commercialization of Higher Education.* Princeton, N.J.: Princeton University Press.

Boston Consulting Group. 2003. *Boston Consulting Group/OSDN Hacker Survey.* Boston: Boston Consulting Group.

Caillaud, Bernard. 2003. "La Propriété Intellectuelle sur les Logiciels." *Propriété Intellectuelle.* Conseil D'Analyse Economique, Rapport 41, pp. 113–71.

Casadesus-Masanell, Ramon and Pankaj Ghemawat. 2003. "Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows." Strategy Unit Working Paper 04-012, Graduate School of Business Administration, Harvard University.

Dessein, Wouter. 2002. "Authority and Communication in Organizations." *Review of Economic Studies.* 69:4, pp. 811–38.

European Commission, Interchange of Data between Administrations. 2001. "Study into the Use of Open Source Software in the Public Sector." June; Available at ⟨http://europa.eu.int/ISPO/ida/jsps/index.jsp?fuseAction=showDocument&documentID=333&parent=chapter&preChapterID=0-17-134⟩.

Farrell, Joseph and Michael L. Katz. 2000. "Innovation, Rent Extraction, and Integration in Systems Markets." *Journal of Industrial Economics.* 48:4, pp. 413–32.

Franke, Nikolaus and Eric von Hippel. 2003. "Satisfying Heterogeneous User Needs via Innovation Tool Kits: The Case of Apache Security Software." *Research Policy.* 32:7, pp. 1199–215.

Gaudeul, Alexandre. 2004. "Competition between Open-Source and Proprietary Software: The $(L^A)T_EX$ case Study." Unpublished working paper, Universities of Toulouse and Southampton.

Graham, Stuart and David C. Mowery. 2003. "Intellectual Property Protection in the Software Industry," in *Patents in the Knowledge-Based Economy: Proceedings of the Science, Technology and Economic Policy Board.* Wesley Cohen and Steven Merrill, eds. Washington, National Academies Press.

Greenwald, Bruce C. 1986. "Adverse Selection in the Labour Market." *Review of Economic Studies.* 53:3, pp. 325–47.

Hahn, Robert W. and Scott J. Wallsten. 2003. "A Review of Bessen and Hunt's Analysis of Software Patents." Unpublished working paper, American Enterprise Institute-Brookings Joint Center for Regulatory Studies.

Hammerly, Jim, Tom Paquin and Susan Walton. 1999. "Freeing the Source: The Story of Mozilla," in *Open Sources: Voices from the Open Source Revolution.* Chris DiBona, Sam Ockman, and Mark Stone, eds. Cambridge, Massachusetts: O'Reilly, pp. 197–206.

Hann, Il-Horn, Jeff Roberts, Sandra Slaughter and Roy Fielding. 2004. "An Empirical Analysis of Economic Returns to Open Source Participation." Unpublished working paper, Carnegie-Mellon University.

Haruvy, Ernan E., Fang Wu and Sujoy Chakravarty. 2003. "Incentives for Developers' Contributions and Product Performance Metrics in Open Source Development: An Empirical Investigation." Unpublished working paper, University of Texas at Dallas.

Heller, Michael and Rebecca Eisenberg. 1998. "Can Patents Deter Innovation? The Anticommons in Biomedical Research." *Science.* 280: 5364, pp. 698–701.

Holmström, Bengt. 1999. "Managerial Incentive Problems: A Dynamic Perspective." *Review of Economic Studies.* 66:1, pp. 169–82.

Jaffe, Adam B. and Josh Lerner. 2001. "Reinventing Public R&D: Patent Law and Technology Transfer from Federal Laboratories." *Rand Journal of Economics.* Spring, 32, pp. 167–98.

Johnson, Justin P. 2002. "Open Source Software: Private Provision of a Public Good." *Journal*

of Economics and Management Strategy. 11:4, pp. 637–62.

Johnson, Justin P. 2004. "Collaboration, Peer Review and Open Source Software." Unpublished working paper, Cornell University.

Kuan, Jennifer. 2001. "Open Source Software as Consumer Integration into Production." Unpublished working paper, Stanford University.

Lach, Saul and Mark Schankerman. 2003. "Incentives and Invention in Universities." Discussion Paper No. 3916, Centre for Economic Policy Research.

Lakhani, Karim and Eric von Hippel. 2003. "How Open Source Software Works: 'Free' User-to-User Assistance." *Research Policy*. 32:6, pp. 923–43.

Lerner, Josh and Jean Tirole. 2002. "Some Simple Economics of Open Source." *Journal of Industrial Economics*. 52:2, pp. 197–234.

Lerner, Josh and Jean Tirole. 2004a. "Efficient Patent Pools." *American Economic Review*. 94:3, pp. 691–711.

Lerner, Josh and Jean Tirole. 2004b. "A Model of Forum Shopping, with Special Reference to Standard Setting Organizations." NBER Working Paper No. 10664.

Lerner, Josh and Jean Tirole. 2005. "The Scope of Open Source Licensing." *Journal of Law, Economics, and Organization*. 21, forthcoming.

Lerner, Josh, Marcin Strojwas and Jean Tirole. 2003. "Cooperative Marketing Agreements between Competitors: Evidence from Patent Pools." NBER Working Paper No. 9680.

"Microsoft at the Power Point." 2003. *Economist*. September 11.

Murray, Fiona and Scott Stern. 2003. "Do Formal Intellectual Property Rights Hinder the Flow of Scientific Knowledge? Evidence from Patent-Paper Pairs." Unpublished working paper, Massachusetts Institute of Technology and Northwestern University.

Open Source Development Labs. 2004. "OSDL Announces First Chinese Member." January 30; Available at ⟨http://www.osdl.org/newsroom/press_releases/2004/2004_01_30_beaverton.html⟩.

President's Information Technology Advisory Committee, Panel on Open Source Software for High End Computing. 2000. "Developing Open Source Software to Advance High End Computing." October; Available at ⟨http://www.hpcc.gov/pubs/pitac/pres-oss-11sep00.pdf⟩

Raymond, Eric. 1999. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Cambridge: O'Reilly.

Saint-Paul, Gilles. 2003. "Growth Effects of Non-Proprietary Innovation." *Journal of the European Economic Association: Papers and Proceedings*. 1:2-3, pp. 429–39.

Schmidt, Klaus and Monika Schnitzer. 2003. "Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market." Discussion Paper No. 3793, Centre for Economic Policy Research.

Shapiro, Carl and Hal R. Varian. 2004. "Linux Adoption in the Public Sector." Unpublished working paper, University of California.

Thursby, Jerry and Marie Thursby. 2003. "Has Licensing Changed Academic Research? Issues of Productivity, Faculty Incentives, and Public Policy." Unpublished working paper, Emory University and Georgia Institute of Technology.

Waldman, Michael. 1984. "Job Assignments, Signaling, and Efficiency." *Rand Journal of Economics*. 15:2, pp. 255–67.