NBER WORKING PAPER SERIES

OPEN SOURCE SOFTWARE: THE NEW INTELLECTUAL PROPERTY PARADIGM

Stephen M. Maurer
Suzanne Scotchmer

Working Paper 12148
http://www.nber.org/papers/w12148

NATIONAL BUREAU OF ECONOMIC RESEARCH
1050 Massachusetts Avenue
Cambridge, MA 02138
March 2006

Both authors can be contacted at 2607 Hearst Ave, MC 7320, Universtiy of California, Berkeley, CA 94720-7320, USA. Emails: smaurer@berkeley.edu, scotch@berkeley.edu. We thank the Toulouse Network on Information Technology for financial support, and Terry Hendershott for thoughtful comments. This paper is forthcoming in T. Hendershott, ed., Handbook of Economics and Information Systems, Amsterdam: Elsevier.

Open Source Software: The New Intellectual Property Paradigm
Stephen M. Maurer and Suzanne Scotchmer
NBER Working Paper No. 12148
March 2006
JEL No. K, L

ABSTRACT

Open source methods for creating software rely on developers who voluntarily reveal code in the expectation that other developers will reciprocate. Open source incentives are distinct from earlier uses of intellectual property, leading to different types of inefficiencies and different biases in R&D investment. Open source style of software development remedies a defect of intellectual property protection, namely, that it does not generally require or encourage disclosure of source code. We review a considerable body of survey evidence and theory that seeks to explain why developers participate in open source collaborations instead of keeping their code proprietary, and evaluates the extent to which open source may improve welfare compared to proprietary development.

Stephen M. Maurer
2607 Hearst Ave, MC 7320
University of California
Berkeley, CA 94720
smaurer@berkeley.edu

Suzanne Scotchmer
Department of Economics and GSPP
2607 Hearst Avenue
University of California
Berkeley, CA 94720-7320
and NBER
scotch@berkeley.edu

OPEN SOURCE SOFTWARE: THE NEW INTELLECTUAL PROPERTY PARADIGM

I. Introduction

II. Incentives for R&D

- **a. Intellectual property and open source**
- **b. Own use**
- **c. Complementary goods and services**
- **d. Signaling**
- **e. Education**
- **f. Achieving network externalities and denying them to others**
- **g. Social Psychology**

III. Stability and Organization

- **a. Who contributes and how much?**
- **b. Who pays?**
- **c. Why licenses?**
- **d. Why leadership?**
- **e. Network effects**

IV. Efficiency Implications

- **a. Disclosure of code**
- **b. Meeting users' needs**
- **c. Deadweight loss and pricing**
- **d. Training and using programmers**
- **e. Free riding**
- **g. Modularity and the organization of the research effort**

V. Open Source and Proprietary Software

- **a. Competition between open source and proprietary software**
- **b. Market segmentation**

VI. Limitations and Extensions

- **a. Limits to open source software**
- **b. Beyond software: drug discovery and geographic information systems**

VII. Conclusion

References

I. Introduction

Open Source Software, which burst on the innovation scene in the mid-1990s, is produced in a completely different way than other commercial products. Workers are usually unpaid; management and direction are limited; and legal restrictions on using the product are modest (Lerner and Tirole, 2004). The open source style of development has various features, but it generally involves software developers making their source code available free-of-charge to end users and improvers, sometimes subject to license restrictions such as GPL and BSD.[2] Developers often work on the code provided by others.

The open source movement is a substantial phenomenon. LINUX runs on 29 million machines (LinuxCounterSite 2005) and Apache runs on 70 million servers (Netcraft 2005). Despite this demonstrated success, survey evidence indicates that the nature of open source activities is changing rapidly (Comino *et al.*, 2005). For example, the survey data of Ghosh *et al*. (2002) show that there is a secular shift from "hobbyist" contributions to "commercial" contributions. It is still unclear to what extent open source will supplant proprietary methods for software development, let alone branch out into other information goods such as pharmaceuticals or geographic data. In this essay, we provide a snapshot of the emerging open source phenomenon, and a discussion of how scholars have tried to make sense of it.

There are several natural questions to ask about the phenomenon. Among them,

- How do open source methods provide sufficient incentive to invest in software, given that users do not pay innovators?
- What is it about computer software, if anything, that calls for a new invention paradigm? Which other inventive activities share these features?
- Does the efficacy of open source depend on licenses (*e.g.* BSD, GPL) and, indirectly, the underlying availability of intellectual property protection?
- Does the market need a coordination mechanism to collectively choose open source over more traditional ways of exploiting intellectual property, and does it always do so when open source is the better choice?
- In what circumstances does open source work better than traditional intellectual property incentives or other funding schemes, such as public sponsorship?

In section II we lay out the various arguments for why open source works as an incentive scheme, and compare it to more traditional uses of intellectual property. In section III we focus on how open source collaborations are organized. In section IV we turn to some of the observable welfare consequences of organizing R&D in this fashion, and in Section V we discuss some of the gaps in what is known about it, and its potential to organize invention in other arenas.

[2] The General Purpose License is a "viral" license that obligates a further developer of the innovation to make it available under the same license. In general, there are no restrictions on use or an obligation to pay, but in some versions there is an obligation for attribution. The Berkeley Software Distribution license originated through UNIX development at the University of California, Berkeley and is not viral. It requires users to give attribution credit to the University but does not prohibit commercial use or development. For a more complete history, see, for example, Weber (2004) and von Hippel (2005).

II. Incentives for R&D

Not all R&D environments call for the same incentive structure. For example, R&D environments differ in the extent to which ideas for investments are scarce or common knowledge, the extent to which disclosure of progress is inevitable or important, the extent to which innovation proceeds cumulatively through the efforts of many contributors, and the extent to which natural restrictions on entry protect innovators.

The open source movement emerged to support an industrial product (software) for which disclosure of code is particularly useful, but not required by intellectual property law. Copyrights for software can be registered without fully revealing the source code, and source code is typically not included in software patents.[3] Source code is typically not disclosed in either regime.

Of course, code can be released under license, but here the nature of the innovative environment matters. If "ideas are scarce" in the sense that each idea for an improvement occurs to a single, random person (Scotchmer 2004, chapter 2), and ideas depend on prior disclosure, then traditional protection through patents and copyrights may cripple inventive activity. This is because rightsholders do not know whom to license and disclose. In the open source regime, full disclosure is automatic, and thus encourages new ideas and re-use of the code by developers who cannot be identified in advance. The surprise is that this can be done while still preserving incentives. In this section we explore those incentives, but first compare open source with the more traditional way of exploiting intellectual property.

Open source communities have been well studied with survey instruments. When asked their motives, survey respondents cite all of the incentives listed below: own use benefits, complementarity with proprietary products sold in the market, signaling, education, and social psychological motives such as altruism or simple enjoyment. In terms of the technical problems that contributors seek to address, Ghosh *et al.* (2002) report that 39.8% are trying to improve the products of other developers. Fewer are trying to realize a good product idea (27%), or trying to solve a problem that could not be solved by proprietary software (29.6%). Among contributors at SourceForge, the top three reasons for participating in open source communities include "work functionality" (33.8%) and "non-work functionality" (29.7%).[4] (Lakhani and Wolf, 2005).

To control for the possibility that these responses are sensitive to the number and phrasing of the questions, Lakhani and Wolf (2005) use factor analysis to group the responses into four classes: Workers who are primarily motivated by education/intellectual stimulation ("Learning and Fun" - 29%), by non-work user needs ("Hobbyists" - 27%), by work-related user needs ("Professionals" - 25%) and by feelings of obligation/community ("Community Believers" - 19%). Significantly, the two user needs categories comprise about one-half of all respondents.

---

[3] See Lemley et al. 2002 at 204-205 (for patents), Samuelson (1984) and U.S. Copyright Circular 61 (for copyrights).

[4] By way of comparison, "own use" responses lag behind "intellectually stimulating" (44.9%) but are well ahead of "Beat Proprietary Software" (11.9%). (Lakhani and Wolf, 2005, Lakhani *et al.*, 2002).

Surveys of the embedded-LINUX community find that most hardware firms release code in order to continue receiving similar donations from others (61.4%), benefit from other participants' efforts to find and fix bugs (59.9%), to be known as a good player in the open source community (58.9%) and because they hope that others will develop their code further (57.7%). Employees working for software companies report broadly similar motives except that they tend to place slightly more emphasis on marketing (*e.g.,* signaling and reputation). This effect is larger for small, young companies than for older and more established firms. (Henkel, 2005b).

A. *Intellectual Property and Open Source*

Because the key feature of open source is that the knowledge (software) is put in the public domain, open source would not perform well as an incentive mechanism in the usual innovation environment where the objective is to prevent imitation. To the contrary, the point of putting the knowledge in the public domain is to *encourage* imitation. As we will see in this section, open source works in environments where the knowledge created (a) is complementary to some *other* good whose profitability is immune to imitation, such as human capital or another proprietary product, or (b) where the motives to invent are intrinsic and have nothing to do with appropriating value.[5]

We begin our discussion by comparing open source and the ordinary use of intellectual property in the two environments where open source is mainly used: where innovations are complementary and where innovation is cumulative.

Following Scotchmer (2004), we distinguish between having ideas, which are random and costless, and creating innovations, which require investment. Let the idea of firm i be summarized by an indicator of its commercial value and its private cost, (v_i,c_i). We describe *complementarity* by the following profit function, for each firm i, where n is the total number of contributions to the open source project, and f is positive, increasing and bounded:

$$v_i f(n) - c_i$$

Complementarity produces a type of network effect: Contributors prefer to contribute to open source projects with many participants. Even though each developer's code is freely usable by others, the commercial value v_i is assumed to survive. The commercial value must therefore be due to some other proprietary product that embeds the software and not directly to the software itself.

To compare open source with patents, assume that if firms keep their contributions proprietary, they cross-license symmetrically at some price ℓ. Then with n contributors, each firm earns licensing revenue $(n-1)\ell$. However, each firm also has licensing obligations in

[5] For discussion of how economists have treated the design of intellectual property in regard to these issues, see Scotchmer (2004). For a broader set of references that also includes legal scholarship, see Menell and Scotchmer (forthcoming).

amount $(n\text{-}1)\ell$, so there is no net burden due to licensing. GPL leads to the same end without imposing the license fees ℓ and without the burden of negotiating.

Notice, however, that participating in such a scheme might not be the best thing from either a private or social point of view if most of the users are not developers so that the open source community confers benefits on nonreciprocating third parties. If such use constitutes the bulk of the social benefits, and if development costs are relatively high, a better incentive scheme would involve royalties. Royalties from third parties may be necessary to cover costs.

Now suppose that the software contributions are *cumulative.* For example, suppose that a set of proprietary products embed a common software product, such as an operating system, which can be improved sequentially by other developers who find bugs or see opportunities for improvement. Suppose further that each improvement increases the quality of the embedded software and that ideas for improvement occur randomly to members of the community to whom the product has been disclosed.

In the cumulative context, traditional IP has the same problem as with complements. If ideas for improvement are scarce, so that a potential improver cannot be identified in advance, then licensing on a one-by-one basis to disclose the prior code will not work very well. Further, if each member makes the same number of contributions as every other, at least in expectation, then their expected receipts and payouts from licensing will be equal for each contributor, and equivalent to making no payments at all under a GPL scheme.

However, the cumulative context is where we see that a GPL license may be useful. A GPL system locks the proprietors into a royalty-free exchange of knowledge. Unlike the case of complements, knowledge exchange is not symmetric because later developers can free ride on earlier ones, but not *vice versa*. Absent GPL, a developer might be tempted to free ride on earlier developers and make his code proprietary so he can impose royalties on later developers. With GPL, he can only revert to royalties by building his code from the ground up. This may be more costly than using prior code and accepting the GPL.

Because the whole point of making code open is to encourage imitation and use, the code itself will not be a profit center for its developer. In the remainder of this section, we discuss why open source works as an incentive system, namely, that the benefits come from own use, other products sold in the market, signaling, and education. We also discuss social psychology motives that claim to transcend traditional economic incentives. Despite broad similarities, these incentives lead to different behaviors and welfare implications.

B. *Own Use*

User innovation has been documented as far back as Victorian times.[6] By the late twentieth century it was ubiquitous: Examples include everything from machine tools to

[6] The practice of creating new information for one's own use is as old as mankind. Scholars have long argued that the apparent perfection of European folklore reflects accumulated interactions (and incremental improvements) of storytellers and audiences. *See, e.g,* Bettelheim (1976). The high ranking of Homer's *Iliad* and *Odyssey* among Western "great books" is an obvious example.

windsurfing (Von Hippel, 2004, Harhoff *et al.*, 2003, Lerner and Tirole, 2002a). The modern open source movement extends this user innovation model to software. (Raymond 1999). The Apache web server collaboration provides a leading example.

Writing code for own use is not entirely about volunteer labor. Eighty-six percent of contributors who participate in open source projects for work reasons are paid for at least some of their work. Not surprisingly, paid contributors spent almost twice as much time as volunteers – 10.3 hours compared to 5.7 hours (Lakhani and Wolf, 2005). Corporations that employ own use strategies to make or improve products that will be sold to mass market products can afford to bear more cost than individuals. At the same time, own use includes substantial non-commercial activity. Lakhani and Wolf (2005) report that 27% of SourceForge contributors write code for "non-work needs." Since 10.9% of all SourceForge listings involve games (Comino *et al.,* 2005), "hobbyist" motives are apparently important.

Own-use incentives may lead to under-provision of code, since the investing party does not appropriate the benefits conferred on third parties. While reciprocity within the open source community may avoid duplication, it does not solve this basic deficiency in incentives. This is also true in the simple model of complementarity that we started with, where every contributor is concerned with own use. The user will not invest if his c_i is larger than his own benefit $v_i f(n)$, even if the cost is smaller than the social benefit $[f(n)-f(n-1)]\Sigma_j v_j$.

C. *Complementary Goods and Services*

We have already stressed that open source incentives will not work if the open source software must itself be a profit center. This is because imitation is its very life blood. Instead, the open source activity must be complementary with something that remains proprietary.

West and Gallagher (2004) refer to open source as "pooled R&D." In particular, companies share code to test software, fix bugs, and to get improvements, feedback, and extensions (Rossi and Bonaccorsi, 2003, 2005), all of which they would otherwise have to do independently with substantial duplicated costs. Contributors can afford to cooperate in this way because the open source software is bundled into different goods and services that are mostly nonrival in the market and allow individual contributors to appropriate benefits. Typical complements include proprietary operating systems; proprietary applications programs; hardware; documentation; distribution through trusted and convenient brand name channels; bundling open source software into convenient, ready-to-use packages; tech support and warranties; custom software services; consulting, education and training; remote services; complete data solutions; making applications more reliable for particular applications or libraries; and organizing fairs and conferences (West and Gallagher, 2004, O'Mahoney, 2003, Varian and Shapiro, 2003, Dahlander, 2004, Ghosh *et al.,* 2002, Harhoff *et al.*, 2003, Henkel, 2005b, Hawkins, 2002, West, 2003, Raymond 1999). Complements are particularly important for server software, desktop/client software, enterprise solutions, IT consulting, IT services, and the embedded software used in appliances like DVDs and cell phones (Ghosh *et al.,* 2002).

Commercial firms are less likely to participate in open source development where competition among them is significant (von Hippel, 2002, Harhoff *et al.,* 2003). Harhoff *et al.*

(2003) present a model with two users who practice in-house innovation, and are imperfect competitors selling products that are enhanced by the open source product.[7] Even though each firm's rival would benefit from a disclosure of its code, the firm may nevertheless disclose it in the hope that a third-party manufacturer will develop the disclosed product still further. Harhoff *et al.* find that, if competition and technology spillovers are both high, and the cost of adopting the manufacturer's improvement is high, there is an equilibrium in which neither rival discloses. However, if the cost of adopting the manufacturer's improvement is less than its benefits, there is also an equilibrium where both disclose, provided competition and spillovers are low enough.

Henkel (2005a) explores a model in which two firms each need two distinct technologies to manufacture their products. If firms cannot share information, each must invest in both technologies. This raises entry costs and makes monopoly more likely. If they choose open source, there are Nash equilibria where each firm specializes in one technology and obtains the other through free riding. Henkel finds that firms with similar technology needs disclose even where competition is strong. However, firms may or may not share information where their needs are different. Henkel finds equilibria in which both firms disclose. Each user performs whatever R&D generates the most value for itself and free-rides otherwise. In this game, heterogeneous needs suppress the temptation to free ride but still produce useful technology spillovers for the entire industry.[8]

Models with competition require two strong assumptions. First, each game assumes payoffs in which the parties can earn (and split) non-zero economic profits. Rivals are protected against competition and entry by some unspecified means. The key issue of open source – appropriability – is introduced as an assumed parameter. Second, the models assume that parties cannot negotiate licenses with one another. In our view, this makes any comparison with intellectual property suspect, since licensing would would also allow the firms to avoid duplication. The authors argue that their licensing assumption is justified by high transactions costs, the legal difficulty of patenting minor (but cumulatively important) innovations, and the alleged weakness of patents and trade secrets (Harhoff *et al.*, 2003, Henkel, 2005a, Von Hippel, 2002).[9]

Empirical studies of the "embedded LINUX" used in proprietary electronic devices like DVDs and cell phones (Henkel, 2005b) provides a market example in which firms can decide between keeping code proprietary and mutual disclosure. In this case, loopholes in the GPL license that give manufacturers the power to keep certain classes of code confidential if they want to.[10] Despite this, roughly half of all industry members (49%) participate in at least limited

[7] Harhoff *et al.* claim that this symmetric duopoly analysis remains qualitatively correct for moderately large or asymmetric oligopolies.

[8] Henkel analogizes this result to a juke box, in which multiple patrons with heterogenous tastes produce a stream of music that benefits everyone.

[9] We note that evidence showing that most licenses earn modest royalties is equally consistent with the proposition that licensing is efficient.

[10] The reason for the loophole is that GPL says that customers are entitled to source code, but confers no broader right on the general public. In the case of embedded LINUX, the customers tend to be a small number of device manufacturers who have good reason to keep their software secret. Other tactics for evading GPL disclosure requirements include releasing drivers as loadable binary modules rather than source code (done at least sometimes by 53.1% of industry respondents); tying revealed software to secret or copyrighted code; releasing code after a

sharing, and this fraction is growing. In general, firms with strong complements tend to release code more readily. For example, 34.5% of hardware companies release code but only 28.6% of software houses do. More surprisingly, small firms reveal substantially more code than large ones. Henkel (2005b) argues that these firms would prefer to develop code in-house, but lack the resources to do so. Small companies frequently rely on open source communities to fix bugs and improve software (Rossi and Bonaccorsi, 2005).

D. *Signaling*

Open source software grew out of an academic environment in which researchers write articles to improve their career prospects. There is extensive anecdotal evidence that high-profile open source programmers can trade on their reputations to gain job offers, shares in commercial companies, and possibly gain access to venture capital (Lerner and Tirole, 2002a, Raymond 1999, Kogut and Metiu, 2000). Hann *et al* (2004) argue that star programmers are an order of magnitude more productive than their peers, so there is much to signal. Roberts *et al.* (2006)'s study of the Apache collaboration shows that signaling benefits grow stronger as workers rise in rank.

A key feature of traditional patent or copyright incentives is that the private value of the right increases with the social value of the contribution. Hence, the prospect of winning an intellectual property right creates a screening mechanism. When a potential inventor has an idea for an innovation, he will compare its cost to a correlate of its social value before deciding to invest.

With signaling incentives, the benefits of the R&D investment are still correlated with the social value of the investment, but less so. Instead of investing in the products with most consumer value, contributors will choose projects that showcase their technical virtuosity (Lerner and Tirole, 2002a). Signaling incentives therefore explain why open source projects tend to involve server operating systems, programming languages, and other applications aimed at sophisticated users (Schmidt and Schnitzer, 2002). Osterloh (2002) argues on this basis that open source collaborations listen to their smartest users while, for example, Microsoft listens to the dumbest. Similarly, Kollock (1999) argues that, as a consequence of signaling, mass market software tends to be under-served, and that open source software ignores "duller, more complex, but no less useful public goods." Signaling is also weak for such useful tasks as reporting bugs, submitting comments, suggesting new functionality, preparing documentation, building easy-to-use interfaces, providing technical support, ensuring backwards compatibility, and writing programs for utilitarian tasks like power management or wizards (Rossi, 2004, Osterloh and Rota, 2004, Schmidt and Schnitzer, 2002). Empirically, the sole Apache rank open to bug hunters has no measurable impact on salary (Hann *et al*., 2004).

Signaling is not confined to individuals. Case studies and survey evidence show that computer companies also participate in open source in order to build reputation, although the number of such companies seems to be small (Dahlander, 2004, Ghosh *et al*., 2002). The Henkel and Tins (2004) study of embedded LINUX finds that 45.4% of manufacturing companies

---

delay of up to 18 months (35.7% of firms); and re-designing software architecture so that functions are moved to the (proprietary) application layer (Henkel (2005b).

disclose code in order to add to technical reputation, and 32.6% disclose to get on mailing lists as a form of marketing.

As in other incentive mechanisms, signaling can create *agency problems*, such as hiding errors or claiming credit for the work of others. Johnson (2004) argues that open source peer reviewers may collude to hide flaws in each others' code. The only empirical evidence of agency problems we have found is due to Gandal and Ferschtman (2005), who point out that signaling incentives are likely to be more significant for licenses such as GPL that ban commercialization than for those such as BSD that allow it. They find that SourceForge contributors submit 2.9 times more lines of code to BSD-type licenses than to GPL-type licenses, and interpret this data as evidence that signaling incentives become less important once contributors deliver enough code to obtain formal credit.

Signaling incentives can also have an impact on code architecture. Schmidt and Schnitzer (2002) speculate that increased modularity makes individual contributions more visible.[11] Assuming that modularity does promote signaling, Weber (2000) argues that open source members may engage in "strategic forking" to become a leader, *i.e.* unnecessarily splitting a collaboration. Dalle and David (2003) similarly hypothesize that programmers gain more reputation by launching new code than by contributing to an existing project; by working on early releases rather than later ones; and by working on frequently called modules (*e.g.* kernels) instead of applications. Dalle and David point out, somewhat optimistically, that these effects can be beneficial if, for example, working on new modules is socially more valuable than extending existing ones.

There is extensive evidence that signaling works. Programmers often receive job offers, stock, and other benefits (Lerner and Tirole, 2002a). Many programmers reportedly believe that being a member of the LINUX community "commands a $10,000 premium on annual wages." (Kogut and Meitu, 2000). Statistical studies by Hann *et al.* (2004) confirm that each promotion above the lowest rank boosts Apache programmer salaries by 13.3 to 29.3%. Similarly, surveys by Bonnacorsi and Rossi (2003, 2005) and Henkel (2005b) confirm that many commercial firms use open source collaborations to find new workers.[12] Finally, Lakhani and Wolf (2005) and Lakhani *et al.* (2002) use factor analysis on their survey to sort respondents into four groups, including "professionals" (25%) who are motivated by signaling ("gaining status") as well as solving work needs. This group is only slightly smaller than their largest group (29%), "learning and fun."

It is harder to know how powerful the signaling incentive is. Lakhani and Wolf (2005) compare reported incentives against the number of hours worked each week, and find that signaling ("reputation") has only about one-third as much impact as the most powerful predictor, creativity. However, this average figure probably obscures the importance of signaling within specific projects. For example, Roberts *et al.* (2006) find that high status Apache volunteers contribute more code than other members.

[11] Baldwin and Clark (2003) disagree, arguing that large numbers of modules dilute the superiority of any one contribution.

[12] This pattern is not universal. Henkel and Tins (2004) report that only 11.5% of hardware manufacturers participate in embedded LINUX in order to find potential employees.

E. *Education*

Due to its emphasis on peer review, open source provides a particularly good vehicle for education (Lakhani and Wolf, 2005). Education incentives are closely aligned with signaling, but focus on objective skills rather than the perceptions of others. For this reason, they avoid the free-rider and agency problems referred to above. Education incentives explain why surveys routinely find roughly one-fifth of all open source volunteers are students.[13]

Education also looms large in the how collaborators report their own motives. In the survey of Lakhani and Wolf (2005), improving skill (41.3%) is the second most common incentive reported by open source volunteers, behind intellectual stimulation (44.9%). Other responses are markedly less important. In the survey of Ghosh *et al.* (2002), 70.5% of respondents report that they are motivated by learning new skills and 67.2% are motivated by sharing knowledge and skills. These are by far the most commonly cited incentives.[14] In the factor analysis of Lakhani and Wolf (2005) and Lakhani *et al.* (2002), the largest group is "learning and fun" at 29%.

F. *Achieving Network Externalities and Denying Them to Others*

Achieving a favorable market position through network externalities is one of the most important strategic goals for companies in the New Economy. Schmidt and Schnitzer (2002) argue that the importance of network effects and switching costs "is largely independent of whether the software is proprietary or open source." Strategies for solidifying a market position can be loosely grouped into four categories.

Achieving Common Standards. Adopting a single, industry-wide open source standard for software fosters a common pool of skilled workers, reduces costs associated with unnecessary versioning, increases the total number of programmers submitting bug reports and extensions, and avoids transactions costs associated with intellectual property such as "patent thickets" and "anticommons" (Rossi and Bonaccorsi, 2005, Lerner and Tirole, 2004, Ghosh *et al*., 2002, Rossi, 2004). Such benefits are particularly attractive for competitively supplied code that is likely to earn low profit margins in any case (West and O'Mahoney, 2005).

Market Penetration. Releasing code in an open source format facilitates customer acceptance by (a) making it impossible for manufacturers to raise prices at a later date, (b)

[13] Lakhani and Wolf (2005) report 19.5% of all open source collaborators are students. Hertel *et al.*, (2003) report a 23% figure while Ghosh *et al.*, (2002) report 21%..

[14] Henkel and Tins (2004) report similar results for the embedded LINUX industry. Among developers who work for software companies, the most commonly reported motivations are getting better personal skills (66.7%), recognition (60%), feedback to boost performance in their current job (56.9%) and demonstrating their skills to future employers (41.7%). Results for developers working for hardware companies are broadly similar. Not surprisingly, education and signaling incentives are even more important for contributors who work for universities, non-profits, and hobbyists. The most common survey responses for this group include obtaining feedback to improve personal skills (75.0%), improving technical reputation (53.6%), and demonstrating skills to future employers (50.0%).

creating a community of developers who will continue to support the code even if the original author abandons it, and (c) releasing information about APIs that dramatically reduces consumer switching costs if companies fail to deliver on their promises (Varian and Shapiro, 2003, Raymond, 1999).

Influencing Standards. Manufacturers often join open source collaborations in order to steer code in directions that favor their own technology. If first mover advantages are strong, companies may even race to reveal code in order to preempt alternatives and gain a permanent advantage (von Hippel, 2002, Harhoff *et al.,* 2003, Varian and Shapiro, 2003, West and Gallagher, 2004). The effect of these dynamics is ambiguous. For every company that hopes to influence code, others may fear that the open source collaboration will be hijacked so that it no longer supports their needs (Ghosh *et al*., 2002, Lerner and Tirole, 2002b). Such fears may persuade would-be participants that it is better not to support a project in the first place. Alternatively, companies may decide that joining open source collaborations is the best way to detect and prevent hijacking.

Blocking Market Dominance By Others. Finally, companies may support open source as a counterweight to dominant proprietary standards such as Windows (Kogut and Metiu, 2000). Individuals may similarly decide that open source is the best way to prevent large corporations from controlling the tools on which their livelihoods depend. O'Mahoney (2003) says, "Informants spoke of their contributions as investment in their future tools: they are creating code that they will never have to pay someone to use again." Kogut and Meitu (2000) report that LINUX contributors are frequently motivated by "a common fear of Microsoft."

Is any of this important? Henkel and Tins (2004) report that only 30% of manufacturers in the embedded LINUX industry reveal their code in hopes of making it an industry standard.

G. *Social Psychology*

So far, we have concentrated on monetizable rewards. But everyday experience confirms that charitable voluntarism can also be a significant force. Experimental economics routinely finds that people contribute more to the provision of public goods than self-interest alone can explain (Kogut and Metiu, 2001). In some cases, voluntarism may also offer practical advantages. For example, blood from volunteer donors contains fewer contaminants (*e.g.* hepatitis) than blood purchased on the open market. (Titmus, 1972).

Social psychology is the natural lens to look at non-monetizable incentives. Following Lakhani and Wolf (2005) we distinguish between extrinsic motives (*i.e.* doing an activity for some separable consequence) and intrinsic motives based on enjoyment or a sense of obligation or community (Lakhani and Wolf, 2005, Osterloh *et al.,* 2003a). *Extrinsic motivations* include the desire for reputation within the open source community, "ego boost," "feelings of personal efficacy" or other similar incentives (Weber, 2000, 2004, Raymond 1999). By contrast, *intrinsic motivations* do not require an audience. They include creative pleasure, sometimes characterized by a "flow state" in which the individual loses track of time (Lakhani and Wolf, 2005), the desire to be part of a team (Roberts, *et al*., 2004), the ability to express creativity, and experiencing satisfaction and accomplishment (Benkler, 2002, Roberts *et al.*, 2006). Intrinsic motivations also

include altruistic incentives, identifying with a particular group, or ideological opposition to proprietary software and software makers[15] (Rossi and Bonaccorsi, 2005, Hertel *et al.*, 2003, Osterloh *et al.*, 2003b, Kollock, 1999).

In choosing projects, extrinsic motivations may lead volunteers to take account of the social benefits they confer, but intrinsic motivations do not. Volunteers will join projects that place a premium on creativity rather than algorithmic solutions; are neither too easy nor too difficult; are challenging, fun, and simple to learn; and are interesting, fast-moving and even glamorous (Dahlander and Magnusson, 2005, Kollock, 1999). Social psychology incentives may also be unusually strong for groups such as teenagers (Benkler, 2002). This may explain why open source contributors are overwhelmingly young, male, and single (Ghosh *et al*., 2002, Lakhani and Wolf 2005). Henkel and Tins (2004) and Hertel *et al*, (2003) report, respectively, that open source contributors are 98% and 96% male.

Commentators claim that social psychology incentives work best when contributors are only asked to contribute small amounts to the public good (Osterloh and Rota, 2004, Osterloh *et al*., 2003b, Baldwin and Clark, 2003). Nevertheless, even weak incentives can sometimes be important. For example, Baldwin and Clark (2003) argue that the decision to reveal previously-written code is a Prisoners Dilemma game. Here, intrinsic motivation may provide sufficient reward to cover the small (but non-trivial) cost of communication.[16] Similarly, Weber (2000) argues that shared culture and norms help to suppress open source forking. Even weak norms may be amplified if network externalities favor a single dominant standard. This can happen if the average programmer favors strong economies of scale, dislikes porting to multiple versions, and wants to minimize conflict (Lerner and Tirole, 2002a).

The significance of social psychological incentives can also change over the life of a project. Osterloh and Rota (2004) and Franck and Jungwirth (2002) argue that social psychological incentives are most significant for young projects where monetary or reputation incentives are comparatively weak. On the other side, Lerner and Tirole (2002a) argue that visibility of early contributions encourages reputation-seekers to join at an early stage. Conversely, social psychology incentives are expected to weaken as the original volunteers lose energy or mature projects become less fast-paced and interesting (Lerner and Tirole, 2002b, Dahlander and Magnusson, 2005). Social psychology motives may also be less important for company-sponsored projects, which usually start with a large mass of mature code that offers little sense of ownership or creative excitement (West and O'Mahoney, 2005). Finally,

[15] The most famous suggestion for obligation/community incentive is due to Raymond (1999), who argued that open source was driven by a post-modern"gift culture" in which social status is determined "not by what you control but by what you give away." This view represents an elaboration of an older anthropology literature in which gift giving creates the "compulsion to return a gift," confers status and power on the giver, fosters kinship-like relations in which each person "takes what they need and gives what they can"; and encourages gift recipients to praise the giver and feel solidarity with the community (Zeitlyn, 2003, Bergquist and Ljungberg, 2001). One problem with Raymond (1999)'s original formulation argument is that it claimed that gift culture was characteristic of a post-scarcity society. As Weber points out, this argument ignores the fact that "time and brainspace of smart, creative people are not abundant" (Weber, 2004, 2000).

[16] Other incentives, including signaling, may also play a role. Alternatively, Baldwin and Clark (2003) argue that open source projects may resemble multistage games without a definite time horizon. This would obviate the problems associated with one-shot Prisoners Dilemma games.

ideologically-motivated volunteers may decide that the best way to accomplish their goals is to start as many new collaborations as possible. If so, their best strategy may be to leave projects as soon as signaling or monetary incentives kick in (Franck and Jungwirth, 2002).

A virtue of enjoyment as an incentive is that free riding does not destroy it, at least in principle (von Hippel, 2002). Open source collaborations based on social psychology incentives therefore escape the "game of chicken" dynamics that cause delay under other incentives. Bitzer *et al*. (2004) explore games in which developers gain utility from a combination of conventional own-use incentives and two social psychology incentives (fun and pleasure from giving gifts). They find that if the value of social psychology incentives exceeds expected costs, open source members start to produce code immediately. The situation is different, however, if volunteers need *both* types of incentives to cover costs. In this new equilibrium, one member develops code immediately while the others wait. Bitzer *et al.* also explore an extended model in which agents have a finite life because they expect to change jobs and/or see their human capital become obsolete. In this richer model, whichever agent expects to receive open source payoffs for the longest time realizes that she cannot win a waiting game and starts to write code immediately. Based on this analysis, Bitzer *et al.* argue that real-world open source development should be swift. They also predict that open source members will often be unusually talented and well-educated, place unusual emphasis on own-use, gift, and play, be patient (have a low discount rate for future benefits) and also young (hence, have long time horizons).

Many observers claim that external incentives can crowd out intrinsic ones, particularly when the new mechanisms are accompanied by monitoring, control, or time pressure (Osterloh and Rota, 2004, Osterloh *et al*., 2003b). For now, there is little empirical evidence of this. Lakhani and Wolf (2005) report that mixing salary with creativity and political goals does not reduce work effort.

Social psychology theorists also predict that intrinsic incentives will decline if there is a widespread perception that third parties are profiting from the community's efforts. Such perceptions are said to make otherwise willing volunteers feel exploited (Osterloh *et al.*, 2003b, Kollock, 1999). Once again, empirical evidence is limited. Putative cures for crowding out include demonstrating that rules exist and are enforced through social pressure; demonstrated transparency and procedural fairness; encouraging self governance and allowing volunteers to choose their own projects; promoting group identity; providing clearly articulated goals; and making sure that contributions are visible so that members know their contributions are being reciprocated (Osterloh *et al.*, 2003b, O'Mahoney, 2003, Kogut and Metiu 2001, Kollock, 1999). One problem with these suggestions is that they are more or less identical to the recommendations that one would expect from an analysis of games based on "own use" or "signaling" rewards. It would be interesting to know whether social psychology literature implies any distinctive predictions.[17]

Survey responses suggest that different social psychology incentives have wildly different strengths. For example, Lakhani and Wolf (2005) and Lakhani *et al* (2002) find that that intellectual stimulation is the most commonly cited incentive (44.9%) among Sourceforge developers. However, other social psychology incentives appear to be less substantial. These

[17] Licenses that block commercialization are also said to be part of this strategy. *See* Section IV, below.

include believing that code should be open (33.1%), feeling an obligation to repay the community (28.6%), and deriving enjoyment from a team enterprise (20.3%). Finally, ideological motives are extremely weak. Only a handful of respondents (11.3%) report that they participate in open source code in order to beat proprietary software and only half of these (5.4%) feel so strongly that they would "never" participate in a closed source project. Ghosh *et al*. (2002) similarly find that social psychology motivations tend to be reported less often than own use, education, or signaling. They find that the most common social psychology motives include participating in a "new form of cooperation" (37.2%), participating in the open source scene (35.5%), believing that software should not be a proprietary good (37.9%), and limiting the power of large software companies (28.9%).

The problem with such self-reported responses is that they may convey politically correct opinions more than actual incentives.[18] When survey data are combined with information on work effort, intrinsic motives decline sharply in importance. For example, Roberts *et al.* (2006) find that intrinsic motivations are a statistically insignificant predictor of how much software Apache volunteers contribute. Similarly, Hertel *et al.* (2003) find that social psychology incentives and identification with the Linux community are poor predictors of how many hours members will work. On the other hand, Lakhani and Wolf (2005) report that a personal sense of creativity has the largest impact on hours worked per week. They find that this impact is twice as large as enjoyment or receiving a salary and three times larger than reputation incentives.[19]

Referring again to the cluster analysis of Lakhani and Wolf (2005), the smallest of the four clusters (19%) consists of respondents who are primarily motivated by obligation or community-based intrinsic motivations. Within this group, large majorities report that open source is their most creative experience (61%) and that they lose track of time (*i.e.* experience "flow states") while programming (73%). Similar majorities either strongly (42%) or somewhat agreed (41%) that the hacker community provided their primary sense of identity.

III. Stability and Organizational Issues

A. *Who contributes, and how much?*

The amount of effort expended by open source volunteers varies widely from individual to individual. Open source projects typically begin with a single programmer making a large investment. (Raymond, 1999). Even after additional volunteers join the collaboration, much of the work continues to be done by small minorities. Ghosh *et al.* (2002) report that 10% of Sourceforge developers create 74% of code. Similarly, Mockus *et al.* (2002) tell us that 15 core developers provide 83% of all Apache contributions while Von Krogh *et al*. (2003) report that

[18]The appeal of "politically correct" responses is particularly evident for embedded LINUX. Despite obvious business motivations, Henkel and Tins (2004) report that more than 90% of the developers claim to be motivated by a desire to give code back to the community. This was true whether the developers worked for hardware companies (93%), software companies (91.7%) or non-profits and universities (92.6%).

[19]Social psychology motivations seem to have little effect on corporate behavior. A survey of 146 Italian open source companies found that ideological statements had no measurable effect in predicting whether a company would contribute to open source programs. The authors conclude that corporations express such sentiments in order to please developers (Rossi and Bonaccorsi 2005).

about one percent of Freenet's members account for fifty percent of all developer e-mails.[20] This disproportionate reliance on small minorities is particularly evident for new code. Bergquist and Ljungberg (2001) report that almost all new functionality is provided by small groups while Von Krogh *et al.* (2003) report that 13% of Freenet developers provide 53% of all new Freenet code. In open source tech support, 2% of tech support providers supplied 50% of all answers (Lakhani and Von Hippel, 2000).

Small contributors are nevertheless important. This is particularly true for bug reporting.[21] In many open source collaborations, these functions are almost entirely performed by volunteers (Bergquist and Ljungberg, 2001). Mockus *et al.* (2002) estimate that 87% of the Apache members who reported bugs submitted just one report. Hann *et al.* (2004) report that many Apache volunteers who report bugs have just one encounter with the project. While less glamorous than creating new code, these activities may be more valuable than creating code in the first place. According to Bessen (2004), testing, debugging, and maintenance account for 82% of software costs.[22]

Finally, firms face special obstacles in trying to convince open source volunteers to form communities around their products. In particular, firms must persuade volunteers that the code has value as an open source project and is not simply being abandoned because it is technically flawed or losing market share. (West and Gallagher, 2004, Lerner and Tirole, 2002b). Perhaps the best way for companies to demonstrate that the code has promise is to make high-profile investments in building an open source collaboration. This can be done by supplying personnel, offering infrastructure like user tool kits, providing awards and other recognition mechanisms for contributors, bundling open source code with company products, and providing coordination functions like recruiting potential contributors, integrating their efforts, building on-line forums and mailing lists, and developing governance structures (Kogut and Meitu, 2000, West, 2003, West and Gallagher, 2004, West and O'Mahoney, 2005).

B. *Who Pays?*

The next big question is who pays for open source. Firms participate in open source communities almost as much as individuals do. Roughly half of all open source workers are directly or indirectly supported by corporations.[23] Ghosh *et al* (2002) report that 54% of respondents were paid for open source work; Lakhani and Wolf (2005) report that 55% of respondents contributed code during work hours; and Hertel *et al* (2003) report that 43% of LINUX kernel developers sometimes or always receive salary for their work.

---

[20] The small size of core groups may be enforced by technology. Mockus *et al.* (2002) argue that cores larger than 12 to 15 members find it almost impossible to coordinate their actions. Once this point is reached, the number of incompatibilities generated by new software quickly becomes unmanageable.

[21] Bug patching represents an intermediate case: The number of people who fix bugs is an order of magnitude larger than those who write new code but an order of magnitude smaller than those who merely report bugs (Mockus *et al.,* 2002, Kogut and Metiu, 2001).

[22] Kogut and Metiu (2001) similarly report that maintenance activities account for 50 to 80% of all software costs. See also Raymond (1999).

[23] Approximately one-third (32%) of the world's 25 largest software companies engage in significant open source activities. IBM reputedly spent $1 billion on open source projects in 2001 (Ghosh *et al.*, 2002).

Such data probably understate the importance of corporate support, since other survey data suggest that paid open source contributors work more hours, are older and more educated, and devote more time to communication and coordination activities. In the data of Lakhani and Wolf (2005), paid workers spend roughly twice as much time on open source projects as unpaid ones. Similarly, Roberts *et al*. (2006) show that salary is an important predictor of effort. Similar arguments can be found in Ghosh *et al*. (2002) and Kogut and Meitu (2000).

C. *Why Licenses?*

Most open source communities assume that restrictive licenses like GPL are beneficial or at least unavoidable. However, the need for licenses is not entirely obvious nor, assuming that licenses are needed, is it clear which restrictions are necessary or desirable.[24] From a welfare standpoint, the best way to ensure use and re-use of software would be to place it in the public domain without any license at all. This strategy would also be simpler to implement than the elaborate licenses that open source actually uses. This section describes five possible reasons why open source licenses might, after all, be necessary. Intriguingly, most can be satisfied with licenses that are significantly less restrictive than GPL.

Symbolism. Dahlander and Magnusson (2005) argue that licenses are sometimes chosen for symbolic reasons. Presumably, the need for symbolism is linked to social psychology incentives that would erode in the presence of private gain, Lakhani and Wolf (2005), or the absence of rules enforcing reciprocity. Kollock (1999). By now, there is reason to be skeptical of these explanations. For example, Kogut and Metiu (2001) argued five years ago that the comparatively restrictive Apache license posed a threat to the "generalized reciprocity that characterizes the community culture." Despite this, Apache has thrived.

Lerner and Tirole (2002b) present a more nuanced argument based on the hypothesis that social psychology incentives and GPL licenses only become necessary when alternative incentives (*e.g.,* ego gratification, signaling, and own-use) are weak. This hypothesis may explain why GPL-style licenses are much less prevalent among projects aimed at developers and system administrators than for games and other applications aimed at end-users. Similarly, the symbolic use of licenses may be particularly tempting when open source collaborations with corporate partners place social psychology incentives under threat. Many commentators argue that ideologically motivated licenses and norms pose formidable barriers for most firms (West, 2003, Dahlander and Magnusson, 2005, Bonnacorsi and Rossi, 2003, Dahlander, 2004). That said, the experience of Red Hat and several other companies shows that GPL licenses do not pose insurmountable barriers to business. (Weber 2004, Osterloh *et al*., 2002).

Protecting Complementary Investments. We stressed above that incentives for participation arise partly through proprietary complements, whether human capital or commercial products. The license may be integral to this, for example by preventing users from

[24] Like most standards, choice of license exhibits strong network effects. open source collaborations that adopt widespread preexisting licenses face fewer legal impediments to sharing and merging code. They also save the substantial costs associate with writing and learning a new license (Lerner and Tirole, 2002b). The fact that three-fourths (72%) of all SourceForge collaborations use GPL suggests that network effects are substantial. In this view, the dominance of GPL is little more than an historical accident – neither inevitable nor optimal.

removing an author's name (Franck and Jungwirth, 2002). Licenses can also shield programmers from potential liability (Bonaccorsi and Rossi, 2003, Gomulkiewicz, 1999).

Preventing Forking and Hijacking. GPL-style blanket prohibitions on commercialization can keep open source collaborations from forking or being diverted in unintended directions ("hijacking"). However, this problem can also be addressed by alternative and less intrusive measures such as giving a trusted leader the exclusive right to decide which changes and extensions become part of the official code (Bonaccorsi and Rossi, 2003, O'Mahoney, 2003), social pressure ("flaming" and "shunning") (Raymond, 1999), trademark (O'Mahoney, 2003), and the use of charisma or astute political skills (Weber, 2000 and Raymond, 1999). Furthermore, it is unclear how much danger forking poses in any case. Despite several historical examples (see Varian and Shapiro 2003 and Lerner and Tirole, 2002a), only 11.6% of embedded LINUX respondents see forking as a threat (Henkel and Tins, 2004). Even if no special steps are taken, Weber (2000) argues that network effects tend to suppress forking in any case.[25]

LINUX provides an instructive example of how measures short of GPL-style prohibitions can be used to prevent forking. Although leaders decide which code carries the trademarked LINUX name, users remain free to develop unbranded versions. From the point of view of consumer sovereignty, this may be better than to prohibit forking altogether.

Ideology. Franck and Jungwirth (2002) argue that licenses reassure activists that their donated labor will not be diverted to commercial ventures that enrich open source leaders.[26] More broadly, activists may adopt GPL-style licenses in order to force programmers to choose between writing non-commercial extensions of code and no code at all. Such tactics presumably increase open source software production while reducing the total stock of open plus-proprietary extensions in the market.

Stabilizing Open Source Against IP Incentives. We argue above that, absent GPL, an opportunistic developer will be tempted to free ride on earlier developers and make his code proprietary so he can impose royalties on later ones. This argument for why GPL may be necessary in the cumulative context is similar to an argument of Hall (2004) and Gamberdella and Hall (2005). They point out that a community member who "cheats" by making his code proprietary gets a discrete jump in income, but only reduces the community's output of publicly available by an infinitesimal amount. This leads to a Prisoner's Dilemma in which each community member might choose proprietary rights even though they would prefer a collective strategy that preserved open source. The game can be stabilized if actors are required to make decisions within large, organized groups. Norms, deference to lead researchers, and GPL-style licenses all serve this function. The net result is to stabilize open source so that proprietary code is no longer an "absorbing state."

[25]Weber argues that forkers necessarily forfeit the benefits of a large community. Thus, they cannot offer followers a built-in audience needed to support signaling incentives or generate bug reports. Community members may also resist forking on the theory that it opens the door to further splintering and balkanization.

[26]Franck and Jungwirth further analogize GPL's restrictions to the "non-distribution" clauses that prevent traditional charities from diverting surpluses to managers or outsiders.

D. *Why Leadership?*

Open source communities are often romanticized as collections of atomistic volunteers who self-organize in minimally hierarchical environments. At the same time, open source leaders like Linus Torvalds have cult status. Clearly, self-organization has its limits. In general, open source collaborations rely on leadership to solve a variety of information, agency, and coordination functions.

Information problems are particularly important in the early stages of open source collaborations, when would-be contributors must decide whether the proposed project is feasible, interesting, and potentially useful. The easiest way for leaders to demonstrate these qualities is to provide working software, even if it is incomplete and flawed (Lerner and Tirole, 2002a).[27] Survey respondents report that the most important leadership functions are providing an initial code base (48.6%), writing code (34.3%) and creating a promise/vision (32.3%) (Lakhani *et al.*, 2002). After an open source collaboration has been established, information problems change. However, there must still be a procedure to decide which contributions should be included in new releases. Trusted leaders play a key role in this process (Lerner and Tirole, 2002a).

Leadership also mitigates agency problems. For example, corporations can commit to keep code in the public domain or to highlight individual contributions by surrendering control to outside leaders (Lerner and Tirole, 2004). Following Max Weber, successful leaders must persuade volunteers that their objectives are congruent and not polluted by ego, commercial incentives, or biases (Weber, 2000).

Finally, leadership accomplishes a variety of coordination functions. For example, leaders let volunteers know which projects are worth supporting and, conversely, which constitute forking or abandonment. More generally, someone must make basic architecture choices (*e.g.* modularity) and coordinate work by volunteers. These functions mitigate the delays and inefficiencies that arise when volunteers interact through decentralized games. (Kogut and Meitu, 2000, Lerner and Tirole, 2002a).

E. *Network Effects*

Proprietary software creates significant network externalities, and it is hard to see why open source software would be any different. (Schmidt and Schnitzer, 2002). For example, there is anecdotal evidence that small declines in the popularity of open source projects can snowball into collapse (Lerner and Tirole, 2002a).

There are several reasons to believe that open source collaborations should exhibit significant network effects. First, Weber (2000) argues that large open source communities have an intrinsic advantage over small ones, since they are more likely to include "outliers who have a high level of interest and surplus resources of time and mindspace." These outliers may reflect variance within a single population of contributors or else the presence of distinct sub-

[27] Leadership is especially important where the open source collaboration is built around donated corporate software. Open source volunteers frequently interpret the corporation's decision to forego intellectual property rights as evidence that the code is worthless (Lerner and Tirole, 2002b).

populations within a given open source collaboration.[28] Second, open source projects need a critical mass of participants. If a competing proprietary system siphons off volunteers and ideas, open source becomes less exciting (Lerner and Tirole, 2002a). Third, some important open source incentives (*e.g.* ego gratification, signaling) scale with the size of the audience (Lerner and Tirole, 2002a). Finally, bug identification and bug fixing also scale with the number of users.

With network externalities, it is natural to expect multiple equilibria and/or a winner-take-all dynamic (Lerner and Tirole, 2002a, Weber, 2000). Surveys of open source projects are consistent with this intuition. Healy and Schussman (2003) observe, "It seems clear that for every successful open source project there are thousands of unsuccessful ones."[29] Indeed, most open source projects are very small scale. Ghosh *et al*. (2002) find that 29% of projects have just one author, only 45% had more than two, and only one percent had more than fifty.[30] Similarly, most open source projects never produce much code.[31] Healy and Schussman comment that "Raymond's image of the bazaar does not capture the fact that the typical project has one developer, no discussion or bug reports, and is not downloaded by anyone." In this environment, agonizing about the choice of license or modular architecture is largely symbolic. Project success may depend on accident as much as quality.

IV. Efficiency Implications

As pointed out above, the open source mechanism does not include a means to appropriate benefits conferred on third parties. We would therefore expect underprovision. Other incentives not based on appropriability (*e.g.,* signaling and education) mitigate this problem but also pose a risk of over-provision. In this section we discuss some of those issues, and take the opportunity to contrast the inefficiencies of open source against the inefficiencies incurred by the more ordinary use of intellectual property.

A. Disclosure of Code

As we have discussed above, a failing of proprietary software is that proprietors almost never make their source code available to users. Open code enhances welfare in several respects. It is easier to adapt and reuse, and therefore may retain its value rather than becoming obsolete (Raymond, 1999). It facilitates finding, diagnosing and fixing bugs (Schmidt and Schnitzer, 2002). And it reduces entry costs for firms that supply customization and support services, increasing the likelihood that such services will be competitively supplied (Hawkins, 2002).

---

[28]One frequently overlooked cost factor is competing uses of the worker time. Hertel *et al.* (2003) report that "tolerance for time losses" is an important predictor of effort.

[29]Provocatively, Madey *et al.* (2005) find a power law distribution for the number of developers involved in open source projects. The reasons for this distribution are obscure.

[30]*See also,* Comino *et al.* (2005), reporting that 80% of SourceForge projects have at most two developers. Ninety-nine percent of projects have 16 or fewer developers. Healy and Schussman (2003) report that the median number of developers in SourceForge databases is one and that projects in the 95th percentile have only five active developers. Ghosh and Prakash (2000) report that 76% of all projects in their survey have just one author and only 2% have more than five.

[31]According to Comino *et al.* (2005), 80% of SourceForge projects show no activity since registration. Healy and Schussman (2003) find little or no programming activity in more than half of all SourceForge projects.

But if these advantages are so substantial, why doesn't the proprietary software industry exploit them for profit? Although formal intellectual property protection does not require disclosure, neither does it prevent it. Given better enforcement (stronger protection), proprietary firms might decide that the protection against copying that comes from keeping the code closed is no longer necessary. From this standpoint, the failings of the proprietary software industry arise from the *weakness* of intellectual property, not from its strength.[32] In fact, proprietary firms already share source code to some extent. Examples include giving developer toolkits to licensees (Bessen, 2004), sharing source code with selected developers (Microsoft's "Shared Source Initiative"), and promoting code-reuse inside Microsoft itself (Lerner and Tirole, 2002a).

B. *Meeting Users' Needs*

We distinguish here between the *incentive* to meet the needs of users, and the *ability* to do so. Obviously own-use incentives are directed to the user's needs, even if not to the needs of third parties. However, a programmer's incentive to signal his proficiency, or to become a better programmer, or to participate in a community of altruistic providers, may not be. Since those incentives are not based on appropriating value from users, there is no need for the innovative activity to track user needs particularly closely.

Lakhani and Wolf (2005) tell us that about 58% of all volunteers are IT professionals. Despite their proficiency at writing code, there is no obvious reason they would do mundane tasks useful to third parties like testing for usability (Lakhani and Wolf, 2005), learning unfamiliar programming languages and architectures (Kogut and Meitu, 2000, Von Krogh *et al.*, 2003), deciphering complex commercial code (West and O'Mahoney, 2005), or meeting the needs of highly specialized audiences such as lawyers or accountants (Schmidt and Schnitzer, 2002). Skilled programmers get most benefit from creating tools for other IT professionals. See West (2003), discussing Internet service providers, Johnson (2002), arguing that open source produces more utilities than end user applications, and Comino *et al.* (2005), reporting that two-thirds of all SourceForge projects involve software languages, systems, internet code, communications and multimedia tools, or scientific software.

However, there is a countervailing argument, namely, that open source communities have closer contact with their users than owners of proprietary software, and therefore have a better *ability* to meet their needs (von Hippel, 2002, Henkel and von Hippel, 2005). Mockus *et al.* (2002) point out that programmers in large proprietary software projects frequently do not know the domain for which they are writing. User feedback is particularly valuable where consumer needs cannot be reduced to a few simple criteria of merit (von Hippel, 2005, Kogut and Metiu, 2000). Such information is even more valuable if workers are "lead users" who understand needs, risks, and eventual market size before manufacturers do. In this view, open source allows

[32] This can be overstated. If intellectual property overrewards proprietary software, then it creates deadweight loss that would better be avoided.

user-developers to play a larger role in developing and extending the products they use (Varian and Shapiro, 2003, Kogut and Metiu, 2000).[33]

C. *Deadweight Loss and Pricing*

Proprietary prices are generally above competitive levels. The proprietary price reduces consumption of the patented good and causes users to shift their use to less preferred substitutes. Proprietary pricing can also lead to costly R&D to invent around patents, and may reduce the incentive for second-generation innovators either directly through royalties or indirectly by forcing them to invent around. (Henkel and von Hippel, 2005). Open source avoids these problems by dispensing with intellectual property. Of course, that leaves the puzzle of where the developers get their rewards, which has been the subject of most of this essay.

D. Training and Using Programmers

Proprietary software firms find it hard to observe the competence of a programmer, and therefore cannot tailor wages to programmers' marginal productivities (Johnson, 2004, Gaudeul, 2004).[34] We therefore expect inferior programmers to self-select into private sector jobs that offer a high average wage. Once hired, low quality programmers also have various incentives to hide errors. If they report bugs, they will likely be asked to fix them[35] (Johnson, 2004). If programmers collude in not reporting each other's failings, firms will find it hard to know whether they have good code (few bugs). This undercuts the value of peer review (Johnson, 2004).

Proprietary firms may be forced into suboptimal architectures to control the agency problems. For example, they may substitute top-down supervision for a more stimulating environment in which programmers can devise their own projects and work in parallel on many projects, recognizing that many should fail (Kogut and Metiu, 2000). A management-intensive way of organizing the development effort reduces worker satisfaction as well as worker effectiveness, and firms must offer a wage premium to overcome it. By contrast, open source programmers can join whichever projects fit their personal interests (Mockus *et al*. 2002; Rossi *et al*, 2004). Most open source incentives tie rewards to the actor's own efforts, and therefore avoid the management problems we have identified for proprietary firms.[36] Other things equal, we expect talented programmers to self-select away from the private sector and into open source.[37]

---

[33] Proprietary firms try to replicate these advantages by using in-house developer networks to appraise and debug software (Kogut and Metiu, 2001). Microsoft's Shared Source Initiative similarly lets select customers view, customize and patch an open version of Windows (Lerner and Tirole, 2004).

[34] Gaudeul (2004) also suggests alternative reasons why wages could be higher than marginal product. These include legal costs associated with enforcing intellectual property, organizational costs associated with setting up and managing a firm, and compensation for restricting the developer's choice of projects.

[35] This result requires the admittedly "strong assumption" that contracts are written in a way that pays programmers a flat wage no matter how many tasks are discovered and assigned *ex post* (Johnson, 2004).

[36] This is clearly *not* true to the extent that corporations pay employees to work on a particular project. Even here, however, open source may have significant advantages to the extent that paid workers also have non-cash incentives and/or must interact with members who have such incentives.

[37] We note that this effect is not unique to open source. Proprietary firms frequently reduce agency problems by adopting incentives – notably prizes and patents – that tie worker rewards to measurable results.

Mockus *et al.* (2002) and Rossi *et al*. (2004) take this argument one step further by arguing that talented programmers also self-select into whichever open source project promises to make the best use of their skills. Perhaps the clearest articulation is due to Benkler (2002), who argues that open source selects "*the best person* to produce a specific component of a project, *all abilities and availabilities to work on the specific module within a specific time frame considered*" (emphasis original). Such claims are almost certainly overstated. While there are surely selection effects, there is no obvious way to aggregate information in order to match programmers efficiently with projects. The same difficulty arises for patents and prizes (Scotchmer 2004, chapter 2).

We can ask the same question for labor practices as we asked for disclosure of source code: If open source practices are best, why don't proprietary firms emulate them? The answer, at least to some extent, is that they do. First, some proprietary firms deliberately foster reputation incentives by attaching programmers' names to code. Compared to open source, however, the results are ambiguous. On the one hand, giving credit attracts and elicits more effort from talented workers. On the other, the strategy increases the risk that star employees will be hired away by competitors. (Lerner and Tirole, 2002a, von Hippel, 2002). Second, many firms try to create work environments that respect motives like reciprocity, altruism and being "part of a team." (Lerner and Tirole, 2002a). Finally, some firms have experimented with decentralization. Microsoft's efforts to have employees challenge each other's ideas are one step in this direction (Kogut and Meitu, 2000).

So far we have treated programmers' self-selection decisions as immutable. This is reasonable for incentives based on complementary goods and services, since programmers who join open source communities in order to build their own businesses will presumably stay there. On the other hand, reputation and signaling incentives are meaningless unless programmers eventually return to the private sector. In practice, reviewing code is expensive, and firms prefer to use open source hierarchies and ranking systems as a proxy for quality (Lerner and Tirole, 2002). Weber (2000) argues that open source rankings are inherently trustworthy because programmers know that the best way to build reputation is to work with high quality collaborators and exclude everyone else.

E. *Free Riding*

If ideas are not scarce – that is, if any good idea is likely to be had and implemented by someone else – it is tempting to let someone else bear the development cost. In this environment, open source programmers who are otherwise willing to write code may wait in hopes that someone else will do the job first. (Raymond, 1999). Users of code who do not contribute are *free riders*.

The ability to free ride can reduce welfare. Scholars have used game theory to explore these results. In general they find equilibria with pure strategies in which some developers invest and others free ride, and other equilibria with mixed strategies in which each developer works

with some probability and development sometimes fails.[38] Communities that play mixed strategies will deliver code more slowly or less reliably than proprietary software companies do. (Johnson, 2002, Baldwin and Clark, 2003, Bitzer *et al.,* 2004).

Patent incentives have the opposite problem. Much of the literature on patent incentives is concerned with racing, given that only the winner will have rights. Depending on the commercial value of the patent, racing may lead to too much investment. (See Scotchmer (2004), chapter 4, for an overview).

G. *Modularity and the organization of the research effort*

The degree to which software is modularized affects its suitability for the open source style of development. This suggests that leaders can make open source collaborations stronger by designing more modular architectures. For example, Benkler (2002) claims that modularity ("granularity") is the key to making open source projects viable. His conjecture is supported by survey data suggesting that own-use programmers spend relatively few hours on open source, and tend to lose interest as soon as their own narrow needs are addressed (Roberts *et al.*, 2006).

Johnson (2002) makes a more systematic investigation of the role of modularity in open source. He considers a game in which each developer can choose to create one module based on his own individual ratio of personal benefit to cost. In this model, each developer invests if the added probability of success provided by his own investment, weighted by his personal valuation of the module, outweighs his own personal cost. The probability that at least one member succeeds depends on the number of members who decide to invest. Johnson finds that adding a programmer to the pool, with randomly drawn cost and benefit for each module, typically increases the likelihood that the code will be supplied. Assuming that the ratio of personal benefit to cost is bounded, he finds that the total number of volunteers who write code in any particular period approaches a limit as their numbers increase. Conversely, free-riding increases. Johnson shows that the total investment is smaller (produces less software) than the effort needed to maximize social value in expectation. Intuitively this result follows from the fact that each programmer's decision to contribute depends only on her own personal benefits, and does not reflect spillovers to other users.

Johnson also investigates how variance in (personal) benefit/cost ratios affect the optimality of open source compared to single-source development. Suppose that each of k modules is necessary for completion of the whole. If a single programmer must create the entire project, the code will not be written unless the ratio of total benefit to total cost over all k modules is greater than one for at least one developer. Apply the same test on a module-by-module basis. In this case, the maximum benefit/cost ratio could be greater than one for each module separately even though the maximum ratio of the totals was less than one and *vice versa*. For this reason, a modularized project can succeed where a non-modularized project fails. Nevertheless, Johnson argues that the success rate of modular projects grows with the number of developers because programmers can self-select into working on the modules for which they are

[38] Open source members clearly understand this logic. Hertel *et al.* (2003) find that Linux kernel members work harder when they believe that their contributions are "highly important" to further progress.

most proficient. The main drawback is that the project may founder if costs are badly distributed among modules.

Baldwin and Clark (2003) explore how modularization fares when different programmers can communicate to avoid redundant work. Since workers share costs, a collective effort with adequate communication is always preferable to coding in isolation. Free riding may increase, however, if communications are so slow that programmers cannot monitor which modules have been already written or if the number of developers exceeds the number of modules.[39] Even in this environment, increasing the number of modules reduces free-riding. Similarly, systems with large potential payoffs will normally attract more workers per module provided that there is enough randomness in outcomes and if costs are small enough.[40] Developers may intentionally duplicate each other's efforts in order to obtain a higher best outcome.

The degree to which programmers can coordinate their efforts, either to avoid duplication or to reinforce efforts to achieve particularly valuable modules, depends on how isolated they are. Ghosh *et al.* (2002) report that 17.4% of open source developers have no regular contact with the rest of the community and 67.9% had regular contact with less than five other members. Only 17.3% had regular contact with more than ten other members. In principle, better and more frequent communication could improve the performance. Corporate support, where it exists, may help fill this gap by supporting the "professional elite" who maintain regular contact with 50 or more developers. Ghosh *et al.* (2002).

V. Open Source and Proprietary Software

Open source limits the market power of proprietary code by providing competition and the threat of entry (Henkel and von Hippel, 2005, Bessen, 2004). There are two possible scenarios in which this can happen. First, open source and proprietary code can compete head-to-head within a single market. Second, they can occupy separate niche markets. Commentators have explored both scenarios.

A. *Competition between open source and proprietary software*

One explanation for why proprietary software can survive in competition with open source products is that users prefer it, either because it is better code or because it is user friendly. Another reason might be that proprietary software becomes entrenched by network effects (Casadesus and Ghemawat, forthcoming).

We have already uncovered several reasons why proprietary firms might have a quality advantage. First, open source incentives probably capture less social value than the patent monopoly does, so that certain projects will not be developed in that community (Schmidt and Schnitzer, 2002). Second, if open source relies on signaling incentives, and a programmer receives no benefit from the unglamorous task of making the code user friendly, no one will bother. Third, we have seen that open source contributors may delay writing or improving code

[39] This may be a realistic concern for very large projects like LINUX or Apache. The problem is mitigated if most members are paid to participate by small group of employers.
[40] See also, Johnson (2002).

in hopes that others will step in instead. Since corporations suffer the opposite coordination problem – they want to be first to market with a good product – we expect them to act faster whenever it is profitable to write good code.

There is some evidence that commercial code has a quality edge over open source, at least for some users. Consulting studies report that the up-front costs of purchasing Windows are approximately offset by lower system and administrator costs compared to LINUX. For now, it is unclear whether the cancellation is exact. Varian and Shapiro (2003) suggest that the lifecycle costs of owning LINUX may be 10 to 15% cheaper than Windows, but remark that this difference is not "striking." We remark, though, that such studies face serious methodological challenges including a) difficulties in comparing non-comparable software, b) poor metrics for reliability and quality, c) sensitivity to IT wages, and d) limited information about how much additional IT effort is needed to support LINUX compared to Windows. (Metagroup, 2005, Cybersource, 2004, Robert Frances Group, 2002).

Mustonen (2003) uses a quality differentiation model to show that low quality open source can sometimes coexist with high quality proprietary products. The model assumes that both the programmer and user markets are segmented. Mustonen finds that high quality programmers self-select into the open source community where they are better rewarded for their skills even though the proprietary sector continues to produce superior products by hiring large numbers of low quality programmers. He also assumes that users with high demand are willing to pay a high price for superior software. Provided that consumers can install software at low cost, Mustonen finds an equilibrium in which consumers who place a low valuation on software always choose open source over proprietary code and the proprietors set prices based on demand from high valuation consumers. If installation costs are high, Mustonen finds that proprietary software dominates for large markets that value high-quality software. Conversely, proprietary firms tend to abandon markets that are small or which place a low value on quality.[41]

Kuan (2001) presents a model where proprietary companies offer different versions of their software to high- and low-quality consumers. Absent perfect price discrimination, she finds that low value consumers always switch to open source. However, the willingness of high value consumers and consumer/programmers to switch to open source is less clear, depending on how many such individuals exist. In general, Kuan finds that the analysis is "very undetermined."

Casadesus and Ghemawat (2006) construct a dynamic model of entrenchment through network effects in which a proprietary product and open source product coexist. Starting from a situation where the proprietary product has a network advantage over the (intrinsically similar) open source product, they ask whether the open source product can supplant it. Although the price of the proprietary product will be higher in each period, the proprietor will choose the path of prices so as to maintain a market advantage through the network advantage. Unless network effects are weak, proprietors never lower prices enough to drive out the open source product entirely. Open source and proprietary products therefore coexist forever.

[41] Bessen (2004) similarly predicts that open source may crowd out proprietary software in cases where markets are small or only modest effort is needed to develop an initial product. Where markets grow rapidly, open source may use this advantage in small markets to gain a first-mover advantage over proprietary competitors.

B. *Market Segmentation*

Even if open source and proprietary code do not compete directly, they may serve different niche markets. Bessen (2004) notes that packaged software like Microsoft Windows accounts for only about 30% of all software spending. The remainder is customized software created by users, often in open source communities. However, creating customized software products is expensive. Bessen argues that, because it is difficult to write an enforceable contract for customized software in advance, proprietary firms must negotiate a price *after* the software package is created. But with a single customer, the customer will have a lot of bargaining power (Aghion and Tirole, 1994). Anticipating this, the proprietary firm will be loathe to invest and would-be customers will be forced to assemble the software from open source modules. Bessen argues that users who place a low value on quality or have highly unique needs will choose open source. Users whose needs can be satisfied with simpler applications will continue to use proprietary products.

Gaudeul (2004) explores how market segmentation can arise from developers' profit-maximizing choices about how best to exploit their code. The developer's first choice is to copyright the software and hire developers to implement it, collecting proprietary profit. This option will not be attractive, though, if wages are too high. The innovator's second choice is to obtain development efforts at zero cost by choosing a GPL license. However, GPL strategies are only feasible if the project has sufficient value for the open source community to participate. The innovator's only other choice is to purchase development efforts on the open market. This can be done by offering developers a BSD license. Gaudeul argues that GPL is usually worse from a welfare point of view, since software is developed with lower probability than it would be in a proprietary regime where corporations do all work immediately or a BSD regime in which programmers race for proprietary rights. However, these differences will likely be minor where development costs are small; furthermore, there may be cases in which rising costs reduce the value of proprietary software faster than they undermine GPL incentives. If so, GPL may sometimes generate more social welfare than copyright incentives "for medium level[s] of c[ost]."[42]

VI. Limitations and Extensions

A. *Limits to open source Software*

Open source incentives do not perform all tasks equally well. For this reason, otherwise useful projects can be limited by open source's weakness in supplying specific components such as documentation or tech support. Henkel and Tins (2004) report that lack of documentation is the most commonly reported obstacle (57.5%) to embedded LINUX.[43] Gamberdella and Hall (2005) similarly argue that open source incentives are poorly suited to downstream tasks like

[42] Gaudeul recognizes that his two-stage model is relatively simple. He adds that more complex models would include wasteful duplication due to racing; GPL developers' ability to earn income by selling complements; and the possible advantages of GPL in multi-stage games where developers continue to add new ideas over time.

[43] Open source sometimes works surprisingly well despite seemingly implausible incentives. For example, Lakhani and Von Hippel (2000) report that open source volunteers sometimes provide tech support in order to gain information about bugs. Weber (2000) argues that open source collaborations can accomplish even mundane tasks if a handful of members have unusual preferences.

assembling knowledge into unified, user-friendly formats. GPL licenses may be counterproductive in this situation if they displace intellectual property rights needed to extract full value from society's investment in knowledge.

Many commentators assume that open source networks are larger than corresponding corporations and, for that reason, draw on more widely scattered ideas and expertise (Gamerdella and Hall, 2005, Benkler, 2002, Kogut and Metiu, 2001).[44] Wide dissemination is said to be particularly important in fast changing or complex products (Dahlender, 2004). As we have seen, however, most open source collaborations are far smaller than corporations. Arguments based on the superior ability of open source to elicit knowledge should therefore be viewed with caution except, perhaps, in the case of Open Science.

Finally, we have seen that open source works best for modularized software. However, it is not clear whether the number of modules is a free parameter. Lerner and Tirole (2002a) speculate that the ability to break projects into modules may be a technical accident peculiar to UNIX that will likely fade as open source moves to other languages. Modularity may also be limited by the ability of a small central group to screen submissions for quality and consistency.

B. *Beyond Software: Drug Discovery, Geographic Information Systems, and Wikipedia*

Unlike previous eras, 20th Century R&D specialized in delivering complex systems that required more man-years of effort than a human lifespan could supply. Examples included airplanes, automobiles, rockets and spacecraft, automobiles, pharmaceuticals, silicon chips, and, most recently large computer programs. Prior to open source, all such projects relied on organizing large numbers of contract researchers within rigid, top-down hierarchies. From Moscow to Los Angeles, the scene was strikingly similar: Hundreds and even thousands of engineers sitting at drafting tables or, more recently, computer terminals. One of the most striking aspects of LINUX is that it is possible to organize at least one form of complex invention – large operating systems – a different way. It is therefore tempting to ask whether the open source model can be extended to other information goods. Because of their high information content, the most natural technologies are on-line reference works (*e.g. Wikipedia*) drug discovery and geographic information systems.

The Internet hosts innumerable blogs and wikis where volunteers collect and process information for a wider audience. There are also more structured sites that invite members to perform defined tasks like counting Martian craters, ranking the most interesting news stories, and even writing encyclopedia entries. (Von Hippel, 2005, Benkler 2002). In many ways, these projects resemble and extend an earlier tradition of large nuclear and particle physics databases that have relied on worldwide networks of volunteer editors since the 1940s. (Maurer, 2003). Like those earlier "big science" projects, the quality of at least some on-line resources seems to be high. Recent peer review tests of the Wikipedia on-line encyclopedia suggests that its accuracy is comparable to its most prestigious IP-supported counterpart, *Encyclopedia Britannica*. Giles (2005). In some ways, blogs and wikis may actually be a more favorable environment than software for open source since each article can be written and enjoyed as a

[44] This is related to our argument above that disclosure shakes loose the "scarce ideas" that might otherwise remain untapped.

free-standing "module." The ability of authors to derive value is largely independent of how well their article interacts with other modules or, indeed, whether other articles are written at all. This is very different from open source, in which volunteers cannot derive value unless they are eventually able to combine their parts to create a single, unified product.

Drug discovery and geographic information systems are much closer to the software model. Weber (2004, 2000) and Burk (2002) suggest that deciphering and annotating the genome might be organized as an open source project. More recent observers have usually been skeptical. Lerner and Tirole (2004) suggest that many biotechnology tasks cannot be broken up into modules and, in any case, are expensive. Furthermore, there may not be enough "sophisticated users who can customize the molecules to their own needs" (Lerner and Tirole, 2004). Similarly, Kogut and Metiu (2001) assert that "A molecule … is not modular" because "changing atoms drastically alters its pharmaceutical properties."[45]

Maurer (2006) argues that these objections assume that drug discovery is an indivisible black box. In fact, there are approximately one dozen distinct subtasks between basic science and the delivery of a completed drug. Proprietary companies routinely break the black box apart by offering different incentives for each specific substep. Examples include buying and selling drug ideas ("external innovation"), offering prizes for recalcitrant chemical engineering problems, and outsourcing pre-clinical and clinical trials to outside entities. There is no obvious reason why open source collaborations could not similarly perform one or more substeps. Scholars have suggested open source collaborations for a wide variety of tasks including basic science, using database (bioinformatics) tools to find the gene sequence "targets" that code for disease, computational design of drugs for specific targets, *in vitro* chemistry and biology experiments designed to validate proposed targets and drugs, and clinical trials (Maurer, 2006, Maurer *et al.*, 2004, Benkler, 2002, Von Hippel, personal communication). While convincing examples of open source biology do not yet exist,[46] it is reasonable to think that incentives based on education, signaling, and ideology should appeal to biologists just as much as they do for computer scientists.

The deeper question is whether open source biology can coexist with the powerful incentives offered by conventional patents (Cohen, 2005). One solution is to create open source biology collaborations in fields where patent incentives are weak, for example tropical disease research (Maurer *et al.*, 2004). More speculatively, much of the risk and expense associated with clinical phase trials involves documentation costs that pharmaceutical companies incur to convince a skeptical FDA that their data is unbiased. This problem is aggravated by outsourcing, which gives contract researchers obvious incentives to suppress and even falsify data to keep test programs alive. As noted by Titmus (1972), non-market solutions avoid these problems by relying on volunteers who have nothing to gain by lying. Since most open source incentives

[45]The existence of "me-too" drugs, in which drug companies change existing patented compounds just enough to avoid patent infringement suggests that molecules may be more "modular" than Kogut and Metiu suspect.

[46] Commentators sometimes argue that bioinformatics software projects and/or science collaborations that adopt open source-like licensing terms deserve the label "Open Source Biology." (Rai, 2005, Boettinger and Burk 2004 (describing HapMap license). The former are indistinguishable from other types of software while the latter invariably turn out to be grant-supported collaborations that have adopted open source-like licenses. Whether such collaborations should be called "open source" is, of course, a matter of semantics. Suffice to say, they do not seem fundamentally different from traditional big science projects dating back to the 1930s.

similarly suppress agency problems, conventional pharmaceutical companies might decide that funding OS clinical trials was a more cost-effective way to convince FDA that their drugs were effective than conventional contract research. In this case, patent incentives would reinforce OS biology instead of competing with it.

Finally, Geographic Information Systems (GIS) present a second industry where open source methods may be acquiring a foothold. Like biology, the technology is highly computerized and depends on users to notice and correct errors. Some government consortia already style themselves "open source GIS" and invite users to submit reports when mapping data turns out to be inaccurate (National Research Council, 2004). In theory, proprietary firms could similarly encourage users to report errors. Open source GIS may, however, have an inherent advantage in fostering social psychology incentives (*e.g.* altruism) among users.

How much further can the open source model be extended? Our discussion suggests several factors. First, where contributors make complementary contributions for their own uses, open source projects are more likely to go forward when each contributor's individual cost is small. This suggests that projects with large, costly, and indivisible modules are disfavored. Examples include projects where volunteers would have to build physical prototypes or conduct physical experiments. Second, we hypothesize that most workers in most industries find incentives like ideology, education, and signaling much weaker than IP rewards. If contributors turn to traditional IP instead of open source, open source collaborations may never acquire the critical mass of workers they need to get started. Instead, we expect open source projects to arise in markets where traditional IP incentives are weak.[47] Weaknesses of traditional IP incentives may occur because users are poor (*e.g.* users of drugs for neglected diseases), because extreme ease of copying undercuts the value of formal IP rights (*e.g.*, software, data), or because licensing is impractical due to market imperfections like high transactions costs or the presence of "experience goods". Finally, we have seen that public domain sharing of innovation makes sense for cumulative innovation environments in which "ideas are scarce," so that the owners and would-be improvers of IP cannot readily find each other. In such systems, GPL-style open source stabilizes the public domain by preventing improvers from taking ideas private. The phenomenon will normally be more important for high risk, cutting-edge research (*e.g.* early stage drug discovery) that puts a high value on clever new ideas compared to well-understood fields where successful development is more or less assured once funds are invested (*e.g.* automobiles).

VI. Conclusion

Open source comprises not one but an entire suite of incentives. In general, each has separate and distinct welfare implications. Furthermore, the importance of, say, "signaling" or "own-use" incentives varies significantly across and even within projects. While generalizations are difficult, most open source incentives reduce agency problems and deadweight loss compared to patents, and accelerate discovery through automatic disclosure. Against these virtues, open source incentives often lead to an under-supply of goods relative to the patent

[47] In principle, policymakers can encourage open source by reducing IP benefits. They should not, however, abolish IP entirely. Given that open source incentives tend to undersupply goods they should not replace IP, although they can certainly supplement it.

system. Open source may also be less responsive to certain users, especially when those users are non-programmers.

Because of undersupply, open source can only be a partial solution: It is not viable, and cannot operate in every environment where patent incentives do. Where it works, however, it is often superior.

References

Aghion, P. and J. Tirole, 1994, The Management of Innovation, Quarterly Journal of Economics, 109, 1185.

Baldwin, C. and K. Clark, 2003, Does Code Architecture Mitigate Free Riding in the Open Source Development Model? Harvard University Business School Mimeo.

Benkler, Y., 2002, Coase's Penguin, or Linux and The Nature of the Firm, Yale Law Journal 112, 369.

Bergquist, M. and J. Ljungberg, 2001, The Power of Gifts: Organizing Social Relationships in Open Source Communities, Information Systems Journal 11, 305.

Bessen, J., 2004, Open Source Software: Free Provision of Complex Public Goods, Boston University Law School Mimeo.

Bettelheim, B., 1976, The Uses of Enchantment: The Meaning and Importance of Fairy Tales, Knopf, New York.

Bitzer, J., and W. Schrettl and P. Schroder, 2004, Intrinsic Motivation in Software Development, Free University of Berlin Discussion Paper 2004/19.

Boetinger, S. and D. Burk, 2004, Open Source Patenting, Journal of International Biotechnology Law, 1, 221.

Bonnacorsi, A. and C. Rossi, 2003, Licensing Schemes in the Production and Distribution of Open Source Software: An Empirical Investigation, Sant' Ana School for Advanced Studies Institute for Informatics and Telematics Mimeo.

Burk, D., 2002, Open Source Genomics, Boston University Journal of Science and Technology Law, 8, 254.

Casadesus-Masanell, R. and P. Ghemawat, 2006, Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows, Management Science (forthcoming).

Cohen, W., 2005, Does Open Source Have Legs? in Hahn, R. (ed.) Intellectual Property Rights in Frontier Industries, AEI-Brookings Press 2005), Washington DC.

Comino, S., F. Manenti and M. Parisi, 2005, From Planning to Mature: On The Determinants of Open Source Take Off, University of Trento Department of Economics Mimeo.

Cybersource, 2004, Linux vs. Windows: Total Cost of Ownership Comparison, (self-published).

Dahlander, L., 2004, Appropriating Returns from Open Innovation Processes: A Multiple Case Study of Small Firms in Open Source Software, Chalmers University of Technology Dept. of Industrial Dynamics Mimeo.

Dahlander, L., and M.G. Magnusson, 2005, Relationships Between Open Source Software Companies and Communities: Observations from Nordic Firms, Research Policy 34, 481.

Dalle, J-M and P.M. David, 2003, The Allocation of Software Development Resources in "Open Source" Production Mode, SIEPR Discussion Paper No. 02-27.

Gambardella, A., and B.Hall, 2005, Proprietary vs. Public Domain Licensing of Software and Research Products, NBER working paper 11120.

Gandal, N. and C. Ferschtman, 2005, Open Source Projects: Output per Contributor and Restrictive Licensing, University of Tel Aviv Mimeo.

Gaudeul, A., 2004, Open Source Software Development Patterns and License Terms, University of Toulouse Mimeo.

Ghosh, R. and V. Prakash, 2000, The Orbiten Free Software Survey, First Monday Issue 5, Number 7.

Ghosh, R., R. Glott, B. Kriger, and G. Robles, 2002, Free/Libre and Open Source Software: Survey and Study, University of Maastricht Institute of Infonomics and Berlecon Research GmbH Mimeo.

Giles, J., 2005, Internet Encyclopaedias Go Head to Head, Nature, 438, 900.

Gomulkiewicz, R.W., 1999, How copyleft uses License Rights to Succeed in the Open Source Software Revolution and the Implications for Article 2B, Houston Law Review 36, 179.

Hall, B., 2004, Incentives for Knowledge Production With Many Producers, Cambridge University ESRC Centre for Business Research Working Paper 292.

Hann, I.-H., J. Roberts, S. Slaughter and R. Fielding, 2004, An Empirical Analysis of Economic Returns to Open Source Participation, Carnegie Mellon University Mimeo.

Harhoff, D., J. Henkel and E. von Hippel, 2003, Profiting from Voluntary Information Spillovers: How Users Benefit by Freely Revealing Their Innovations, Research Policy 32, 1753.

Hawkins, R.E., 2004, The economics of Open Source Software for a Competitive Firm: Why Give it Away for Free? Netnomics 6, 103.

Henkel, J., 2004, Patterns of Free Revealing – Balancing Code Sharing and Protection in Commercial Open Source Development, University of Munich Institute for Innovation Research, Technology Management, and Entrepreneurship Mimeo.

Henkel, J., 2005a, The Jukebox Mode of Innovation – A Model of Commercial Open Source Development, Technische Universitat Munich Mimeo.

Henkel, J., 2005b, Selective Revealing in Open Innovation Processes: The Case of Embedded Linux, Technische Universitat Munich Mimeo.

Henkel, J. and M. Tins, 2004, Munich/MIT Survey: Development of Embedded Linux, University of Munich Institute for Innovation Research, Technology Management, and Entrepreneurship Mimeo.

Henkel, J. and E. von Hippel, 2005, Welfare Implications of User Innovation, Journal of Technology Transfer 30, 73.

Hertel, G., S. Niedner, and S. Herrmann, 2003, Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributions to the Linux Kernel, Research Policy, 32, 1159.

von Hippel, E., 2002, Open Source Projects as Horizontal Innovation Networks – By and For Users, MIT Sloan School of Management Working Paper No. 4366-02.

von Hippel, E., 2004, Democratizing Innovation, MIT, New York and London.

Johnson, J.P., 2002, Open Source Software: Private Provision of a Public Good, Journal of Economics and Management Strategy 24, 637.

Johnson, J.P., 2004, Collaboration, Peer Review and Open Source Software, Cornell University Johnson School of Management Mimeo.

Kogut, B., and A. Metiu, 2000, The Emergence of E-Innovation: Insights from Open Source Software Development, University of Pennsylvania Wharton School Reginald H. Jones Center Working Paper WP 00-11.

Kogut, B., and A. Metiu, 2001, Open-Source Software Development and Distributed Innovation, Oxford Review of Economic Policy 17, 248.

Kollock, P., The Economics of Online Cooperation: Gifts and Public Goods in Cyberspace, 1999, in M. Smith and P. Kollock, Communities in Cyberspace, Routledge, London.

von Krogh, G., S. Spaeth, and K. Lakhani, 2003, Community, Joining, and Specialization in Open Source Software Innovation: A Case Study, Research Policy 32:1217.

Kuan, J., 2001, Open Source Software as Consumer Integration into Production, University of California Haas School of Business.

Lakhani, K and E. von Hippel, 2000, How Open Source Software Works: "Free" User-to-User Assistance, MIT Sloan School of Management Working Paper No. 4117.

Lakhani, K., R. Wolf, J. Bates, and C. DiBona, 2002, The Boston Consulting Group Hacker Survey Release 0.73 (self-published).

Lakhani, K. and R. Wolf, 2005, Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, in J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) Perspectives in Free and Open Source Software, MIT, Cambridge and London.

Lemley, M., P. Menell, R. Merges and P. Samuelson 2002, Software and Internet Law (2d ed.), Aspen Law and Business, New York.

Lerner, J. and J. Tirole, 2002a, Some Simple Economics of Open Source, Journal of Industrial Economics, 52, 197.

Lerner, J. and J. Tirole, 2002b, The Scope of Open Source Licensing, Journal of Law, Economics, and Organization 21, 20.

Lerner, J. and Tirole, J., 2004, The Economics of Technology Sharing: Open Source and Beyond, NBER Working Paper 10956.

The Linux Counter, http://counter.li.org/estimates.php (self-published).

Madey, G., V. Freeh, and R. Tynan, 2005, Understanding Open Source as a Self-Organizing Process, University of Notre Dame Department of Computer Science and Engineering Mimeo.

Maurer, S., 2003, New Institutions for Doing Science: From Databases to Open Source Biology, European Policy for Intellectual Property Conference Mimeo.

Maurer, S., 2006, Bill Gates' Other Business – Choosing the Right Investment Strategy for Tropical Disease Research, WHO Bulletin (forthcoming).

Maurer, S., A. Rai, and A. Sali, 2004, Finding Cures for Tropical Disease: Is Open Source an Answer? Public Library of Science: Medicine, 1, 56.

Menell, P. and S. Scotchmer, 2005, Intellectual Property, forthcoming, M. Polinsky and S. Shavell (eds.) Handbook of Law and Economics, Elsevier, Amsterdam.

Metagroup, 2005, File, Web, and Database Server Administration (self-published).

Mockus, A., R.T. Fielding, and J.D. Herbsleb, 2002, Two Case Studies of Open Source Software Development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology, 11, 309.

Mustonen, M., 2003, Copyleft – The Economics of Linux and Other Open Source Software, Information Economics and Policy 15, 99.

National Research Council, 2004, Licensing Geographic Data and Services, National Academy Press: Washington DC.

Netcraft, 2005, Web Server Survey Turns Ten," (self-published).

O'Mahoney, S., 2003, Guarding the Commons: How Community Managed Software Projects Protect Their Work, Research Policy 32, 1179.

Osterloh, M., 2002, Open Source Software: New Rules for the Market Economy, University of Zurich Institute for Research in Business Administration Mimeo.

Osterloh, M. and S. Rota, 2004, Trust and Community in Open Source Software Production, *Analyse und Kritik: Zeitschift fur Sozialtheorie* 26:279.

Osterloh, M., S. Rota and B. Kuster, 2003a, Trust and Commerce in Open Source – A Contradiction? in O. Petrovic, M. Fallenbock, Ch. Kittle, and M. Ksela (eds.), Trust in the Network Economy, 129, Springer: Vienna.

Osterloh, M., S. Rota, and B. Kuster, 2003b, Open Source Software Production: Climbing on the Shoulders of Giants," University of Zurich Institute for Research in Business Administration Mimeo.

Rai, A., 2005, Open and Collaborative Research: A New Model for Biomedicine, in Hahn, R. (ed.) Intellectual Property Rights in Frontier Industries, AEI-Brookings Press 2005), Washington DC.

Raymond, E.S., 1999, The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly, Sebastopol CA.

Robert Frances Group, 2002, Total Cost of Ownership for Linux in the Enterprise (self-published).

Roberts, J., I.-H. Hann, and S. Slaughter, 2006, Understanding the Motivations, Participation and Performance of Open Source Software Developers: a Longitudinal Study of the Apache Projects, Management Science, (forthcoming).

Rossi, M.A., 2004, Decoding the 'Free/Open Source (F/open source) Software Puzzle': A Survey of Theoretical and Empirical Contributions, University of Sienna Dept. of Political Economy Working Paper No. 424.

Rossi, C. and A. Bonaccorsi, 2005, Intrinsic vs. Extrinsic Incentives in Profit-Oriented Firms Supplying Open Source Products and Services, First Monday, Issue 10, No. 5.

Samuelson, P., 1984, CONTU Revisited: The Case against Copyright Protection in Machine-Readable Form, Duke Law Review 663.

Schmidt, K. and M. Schnitzer, 2004, Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market, University of Munich Dept. of Economics Mimeo.

Scotchmer, S., 1991, Standing on the Shoulders of Giants: Cumulative Research and the Patent Law, Journal of Economic Perspectives 5, 29.

Scotchmer, S., 2004, Ideas and Incentives, MIT, Cambridge and London.

Titmuss, R.M., 1972, The Gift Relationship: From Human Blood to Social Policy, Vintage, New York.

United States Copyright Office, 2002, Circular 61: Copyright Registration for Computer Programs, U. S. Government Printing Office, Washington, D.C.

Varian, H.R. and C. Shapiro, 2003, Linux Adoption in the Public Sector: An Economic Analysis, University of California Haas School of Business Mimeo.

Weber, S., 2000, The Political Economy of Open Source Software, Berkeley Roundtable on the International Economy Working Paper 140.

Weber, S., 2004, The Success of Open Source, Harvard University Press, Cambridge and London.

West, J., 2003, How Open is Open Enough? Melding Proprietary and Open Source Platform Strategies, Research Policy, 32:1259.

West, J. and S. Gallagher, 2004, Key Challenges of Open Innovation: Lessons from Open Source Software," San Jose State College of Business Mimeo.

West, J. and S. O'Mahoney, 2005, Contrasting Community Building in Sponsored and Community Founded Open Source Projects," in IEEE Proceedings of the 38th Annual Hawaii International Conference on System Sciences, IEEE Computer Society Press, Los Alamitos CA.

Zeitlyn, D., 2003, Gift Economies in the Development of Open Source Software: Anthropological Reflections, Research Policy 32, 1287.