Economics of Open Source Software *

Justin Pappas Johnson

May 17, 2001

Abstract

A simple model of open source software (as typified by the Linux operating system) is presented. Individual user-programmers decide whether to invest their valuable time and effort to develop a software application that will become a public good if so developed. Open source code potentially allows the entire Internet community to use its combined programming knowledge, creativity and expertise. On the other hand, the lack of a profit motive can result in free riding by individuals and, consequently, unrealized developments. Both the level and distribution of open source development effort are generally inefficient. The benefits and drawbacks of open source versus profit-driven development are presented. The effect of changing the population size of user-programmers is considered; finite and asymptotic results are given. Whether the amount of development will increase when applications have a "modular structure" depends on whether the developer base exceeds a critical size or not. Explanations of several stylized facts about open source software development are given, including why certain useful programs don't get written. Other issues are also explored.

*This paper is an extension of a chapter from my 1999 M.I.T. Ph.D. dissertation. I thank Daron Acemoglu, Travis Broughton, Jonathan Dworak, Frank Fisher, David P. Myatt, two anonymous referees and a coeditor for helpful comments and advice. I especially thank Glenn Ellison for his extensive and concise remarks.

1 Introduction

As of late 2000, studies by Netcraft and E-Soft suggest that the web server Apache powers about 59% of all web pages.[1] In 1998, the operating system of choice on 17% of all new commercial servers was Linux.[2] This is rather striking because both Apache and Linux are now, and have always been, freely available for all. Moreover, their inventors took steps to ensure that their work would always be available at no cost to everyone.

There is a myriad of examples of free software, including Perl and PHP, premier scripting languages for the worldwide web.[3] Much of this software was written in a decentralized fashion by a large number of individual programmers scattered across the world. The sum of these efforts has produced an impressive collection of useful, reliable, and free software.[4]

Such software is commonly referred to as open source software. The source code of a program is the sequence of actual typed common-language words entered by the programmer. These commands constitute the logical structure of the program. When the source code of a particular application is available to all it is said that the source code is open. A competent programmer who has the source code of a program can, given time, figure out exactly how the program works. He or she can modify the program to suit his or her own preferences, correct bugs in the program, or use the components of the program to build a new or extended application. This ability to use ones own programming skills to alter the performance of a pre-existing application can be of considerable value to a serious programmer.

The source code of most programs that one buys is already compiled to run on a particular operating system. Compiled software is binary code that speaks to the components of a computer system. It can be difficult to invert a compiled program to obtain the underlying source code.[5] Also, most firms restrict the rights of end users to modify software written by the firm. As such, most software cannot be usefully modified by anyone other than the original developer. Software for which the source code is not

[1] See `www.netcraft.com/survey` and `www.securityspace.com` for details. The methodologies are somewhat controversial. In particular, servers behind firewalls are not counted.

[2] *Red Herring*, June 1999.

[3] See `www.perl.org` and `www.php.net`

[4] See `sourceforge.org`, `freshmeat.net` and `opensource.org` to better appreciate the sheer scope and depth of open source activity.

[5] The difficulty of decompiling an executable depends on several factors including the language in which the program is written. Even when a program can be decompiled, the generated source code may not match the original code. As a result, it may be difficult to usefully work with generated source code.

generally available is called closed source software.

Computer and software companies are acknowledging the open source movement. In 1998, Netscape (now owned by AOL) opened the source code of its web browser.[6] Sun Microsystems has released the source code of its StarOffice program.[7] Interestingly, StarOffice has been released under the strong terms of the GNU General Public License, which is discussed below. IBM released an open source version of its popular AFS filesystem.[8] IBM has also announced that it will support and market the Red Hat version of the open source Linux operating system (*Red Herring*, February 18, 1999), and also sponsored a three-day open source conference in New York City in December 1999 (*Linux Today*, November 3, 1999). There are many more examples.

In section 2, open source software development is modeled as the private provision of a public good. Section 3 examines the influence of the size of the developer base on welfare, development probability and the distribution of effort and costs. Both finite and asymptotic results are presented.

In section 4, the open source model is compared to a traditional closed source (or profit-driven) model of software development. It is shown that neither system coincides with a constrained social optimum. While the open source paradigm exhibits both inefficient levels and distribution of development it benefits from the fact that individuals know their own preferences better than a firm does and also from the fact that a greater skill set (that belonging to the community of programmers as a whole) can be exploited. The closed source paradigm considers the aggregate enjoyment that consumers will glean from a program, which free-riding open source developers ignore.

In section 5, several stylized facts are explained in the context of the model. In particular, it is argued that a reason the open source community has been able to build immensely complex software objects, such as operating systems, yet arguably been less successful in building other useful applications, such as word processors of quality comparable to proprietary versions, is that a natural correlation between human capital and production technology leads those most able to build applications to build ones that are most useful in their own work.

The importance of the potential for incremental development of an open source application is addressed. In agreement with received wisdom in the

[6] See `www.mozilla.org`.

[7] See `openoffice.org`, for example, or the homepage of Sun Microsystems.

[8] See `www.openafs.org` for details. Technically, IBM has forked the AFS code into an open source version and a proprietary version.

open source community, it is shown that the possibility of incremental improvement is valuable when the developer base is large but that incremental development leads to less development when the developer base is small.

Also in section 5, the stylized fact that open source applications tend to be less complete than their proprietary counterparts is considered. It is shown that this is a natural consequence of profit maximization when development costs are highly correlated across tasks and when reservation prices are additive across different program features.

Before addressing these issues, a very brief discussion of the legal aspect of open source software is in order. In particular, open source licenses will be discussed. Also, the related literature is discussed briefly.

1.1 Open Source Software and Open Source Licenses

The source code of open source software is freely available. However, open source software is more than software for which the source code is available. Open source programs are distributed under very precise licensing agreements. There are many such licenses, only one of which will be discussed in the interest of saving space.

1.1.1 The GNU General Public License

One of the most common and stringent of all open source licenses is the GNU General Public License (GPL). Most of the software mentioned so far is distributed under the GPL license. The GPL grants specific legal rights and responsibilities to those who use and modify products licensed under the GPL.

The GPL does not prohibit charging a positive price for a program covered by the license. However, it grants customers the right to obtain the source code. Moreover, anyone who lawfully obtains a program covered by the GPL automatically inherits the full rights to use, copy, modify or distribute the program and source code in any manner desired, subject only to the terms of the GPL itself (Stallman 1996). Thus, legally, all source code that incorporates GPL source code becomes open source code itself. Thus, while individuals are free to modify programs covered by the GPL, such modifications must be distributed under the terms of the License itself if they are to be distributed at all. Moreover, modifications or redistributions of such open source software must make the terms of the License apparent to others who might obtain the software.

1.2 Related Literature

Open source software development is modeled as the private provision of a public good. Such models of public good provision have been studied by many people, including Chamberlin (1974), Palfrey and Rosenthal (1984), Bergstrom, Blume, and Varian (1986), and Bliss and Nalebuff (1984). The present paper differs from the existing literature both by the focus of the basic model and by the fact that the model is extended to address issues of relevance to the open source development regime.

In particular, the model considers not only whether a project will be successful, but also whether the composition of effort supplied is efficient. Further, the distribution of redundant effort is considered, since much open source development occurs with little or no coordination. Also, the free-riding open source regime is compared both to a social planner's solution and to the solution of a profit-maximizing firm. Finally, the basic model places strong emphasis on a number of asymptotic results (although other papers, for example Chamberlin (1974), do address asymptotic issues).

The extensions of the basic model also distinguish the current paper from previous work. For example, in section 5 the importance of being able to make small contributions is related to the total size of the pool of potential participants.

Lerner and Tirole (2000) explore the economics of open source software as well. Their work differs from the present paper in focus. Their primary point is that labor economics, especially the literature on career concerns, provides a useful framework for understanding some aspects of the open source phenomenon. In contrast, the theory of public goods is central to the present analysis.

2 A Model of Open Source Software

Consider the following simultaneous-move game. There are n user-developers in the Internet community. Each knows that an enhancement of a pre-existing software application, the source code of which is open, can potentially be developed. Developing the enhancement of the software takes time, effort, and ingenuity. These costs are summarized for each agent by his or her privately known cost of development c_i.

Each agent independently decides whether to develop the new application. Any agent i who chooses to develop bears the cost c_i. As long as at least one agent so chooses, the development will occur. Any developed software can be freely provided over the Internet to the other user-developers

and will be so provided if developed (perhaps because the terms of the open source contract vastly restrict a developing agent's ability to profit).

If the enhancement is developed all agents receive their own privately known valuations v_i. If the software is not developed all payoffs equal zero.

Suppose that all agents' costs and valuations are independent, identical draws from the joint distribution function $G(c, v)$, with support on the finite rectangle defined by $\{(c, v) : c_L \leq c \leq c_H,\ v_L \leq v \leq v_H\}$ where $c_L > 0$ and $v_H \geq 0$. Assume this is a smooth function.

The first object of analysis is the optimal response of any agent i to the strategies of the other agents. Suppose that the agent believes that the probability that the development will take place if he or she does not innovate is π^i. A strategy for this agent is a decision to develop with some probability, conditional on his or her own realized cost and valuation, and given his or her beliefs about what other agents are doing (summarized by the value π^i). More precisely, a strategy for each agent i is a function ψ^i taking each of that agent's potential value-cost pairs (v_i, c_i) into the unit interval $[0, 1]$.

An equilibrium of this game is a set of strategies and beliefs $\left\{\left(\psi^i, \pi^i\right)\right\}_{i=1}^n$ such that each strategy is optimal given that agent's beliefs, and such that each agent's beliefs are consistent with the strategies of the other agents. That is, Bayesian Nash Equilibria are considered.

Optimality for agent i requires that ψ^i solve

$$\max_{\psi \in [0,1]} \left[v_i(\pi^i + \psi - \pi^i\psi) - c_i\psi\right]$$

Consistency requires that

$$\pi^i = \Pr\{\text{At least one agent } j \neq i \text{ chooses to develop}\}$$

where the probability on the right is computed using the underlying distribution $G(c, v)$ and the strategies of agents $j \neq i$.

An agent optimally chooses to develop the program with probability one if

$$v_i - c_i > \pi^i v_i$$

which can be rearranged to yield

$$\frac{v_i}{c_i} > \frac{1}{1 - \pi^i} \tag{1}$$

so it is clear that the optimal response is to invest in development with probability one, so that $\psi^i = 1$, if the value-to-cost ratio is sufficiently

high. When inequality (1) is reversed it is optimal never to develop, so that $\psi^i = 0$. When (1) is exactly satisfied, the agent is indifferent among all personal development probabilities. Given the smoothness of $G(c, v)$, this is a measure zero event. Since consistency implies that equilibrium beliefs are invariant to such events it is without loss of generality that one can assume agents who are indifferent choose to develop.

Throughout this entire paper only symmetric equilibria are considered. Let $\hat{q}$ denote the critical value-to-cost ratio. Also, let $F(q)$ be the induced distribution of the value-to-cost quotients;[9] that is,

$$F(q) = \Pr\left\{\frac{v}{c} < q\right\}$$

Denote the upper bound of the support by

$$q_H = \frac{v_H}{c_L} < \infty$$

It will also be convenient to define

$$\gamma = \Pr\{\text{No agent develops}\} = F(\hat{q})^n$$

Given these definitions, the probability (from an individual agent's perspective) that none of the remaining agents develops is

$$1 - \pi = F(\hat{q})^{n-1} = \gamma^{\frac{n-1}{n}}$$

where π is the common value of π^i. Hence one can determine from the agent's optimality condition that he will only be indifferent between developing and not when

$$q_i = \frac{1}{1-\pi} = \gamma^{\frac{1-n}{n}}$$

which of course must equal $\hat{q}$. Given this, inspection of the definition of γ (i.e. the probability that no agent develops) reveals that it is an equilibrium value if

$$\gamma = F\left[\gamma^{\frac{1-n}{n}}\right]^n \tag{2}$$

which has a unique solution unless the law F places no mass above 1, in which case there is no solution to this equation and the unique equilibrium exhibits no development. To avoid this boring case, assume that $F(1) < 1$,

[9]The value-to-cost distribution is not taken as primitive because later the performance of a closed source system will be compared to that of an open source one. To do so the joint distribution of v and c will be needed.

or equivalently that $q_H > 1$. Under this assumption, the above condition is both necessary and sufficient for γ to be an equilibrium value.

The straightforward manner in which the Bayesian Nash Equilibrium[10] can be computed in the basic model has been explained. In the following sections a number of different issues are addressed in the context of the basic model.

2.1 Choice of Model

Before proceeding, a few comments on the choice of the model detailed above are in order. In particular, one might wonder whether the payoffs are appropriately modelled, and whether a static model is preferable to a dynamic one.

Some developers of open source projects in the real world are paid by their employers to spend part of their time working on such projects. This fits within the model, since those same employers are often using the software as an internal tool, not as a product that is sold. That is, the return to the firm of having employees work on the software does not derive from proprietary control and sale of the software.

Another issue is that some user-developers might prefer to develop the software rather than have someone else do it. These people could be trying to signal how clever they are, perhaps out of vanity or the desire to obtain a better job in the future. In fact, the present model captures some of this quite easily. One must merely reinterpret the cost of development to be the net cost, after factoring in other gains beyond mere direct utility from use of the program. Of course, this does require that if multiple agents develop, they are each able to receive these extra cost savings.

If such is not the case, an alternative to the present framework is a tournament model in which developers race to be the first to develop in order to prove their abilities. If this force is dominant, a tournament model might be more appropriate than the model of private provision given above. While difficult to argue cogently, casual examination of the open source movement seems to suggest that a private provision of public goods model is more appropriate than a tournament model. It is a common to hear a lament such as, "It would be great if someone finally could expand the capabilities of this software."

There are several reasons to employ a static model instead of a dynamic one. Clearly, both approaches capture much the same concept; the current

[10] As mentioned previously, throughout this entire paper only symmetric equilibria are considered.

approach allows one to discuss whether an innovation occurs or not, while a dynamic model in the spirit of Bliss and Nalebuff (1984) addresses the issue of delay. It turns out that the present model can be solved easily and in closed form. The results are therefore easy to interpret in terms of model parameters. Also, the current model can be extended easily in new directions, for example to look at the importance of incremental improvement.

Of course, there are strong limitations of the static approach. In particular, such a model cannot capture adequately the incentives of an initial programmer to release her work under an open source license. Furthermore, it strongly relies on the emergence of equilibrium beliefs. Also, a static approach naturally precludes study of how an open source development community acts over time as various improvements and new opportunities for development arrive.

Finally, an alternative to the present framework is a game in which agents have perfect information about the values and costs of all users. Assuming agents were identical, the symmetric equilibrium would exhibit mixed strategies. This approach would then resemble, for example, the work of Dixit and Shapiro (1986), which considers industry entry and exit.

3 The Number of User-Developers

Some open source projects have a greater number of potential developers in the community than others do. Reasons for this include differing awareness about projects and heterogeneity in the underlying value and cost distribution.[11] The number of users in the community influences the equilibrium probability of development, the amount of redundant development effort, and social welfare more generally. Here the influence of the population size on the open source environment is investigated. Both finite and asymptotic results are given.

Finite results are presented first. It is the case that increasing the size of the developer base can lead to a lower overall probability of development. However, any decrease cannot be very large and in any event all agents prefer having more potential developers.

Suppose that the number of user-developers increases. If individuals continued to use their original threshold rule, then clearly development would be more likely. However, when more individuals are present, the incentive to free ride is raised, and any individual will be less likely to develop

[11] For example, it is to be expected the the underlying distributions should in truth be conditioned on the primary field of expertise of the user-developers.

the application herself in equilibrium. Whether the overall probability of development falls or rises as a result of including more agents is therefore ambiguous.[12]

While the movement of the development probability is ambiguous, the likelihood π_n that one of the other $n-1$ agents develops must increase (where subscripts are now used to denote the equilibrium values for a given population size n). This must be true because if the probability of one of the other agents developing were to fall with a growth in population, then each agent would optimally choose to develop more frequently. This would be a contradiction, since if each agent develops more frequently, the probability of at least one agent developing in any subset of agents must also increase.

Lemma 1 *The equilibrium probability that one of the first $n-1$ agents develops is increasing in n. That is, π_n is increasing in n. Also, $\hat{q}_n$ is increasing.*

Proof: Recall that an individual agent i is indifferent between developing and not when $q_i = (1-\pi_n)^{-1}$.

In equilibrium, it is the case that

$$\pi_n = 1 - F\left(\hat{q}\right)^{n-1} = 1 - F\left[\frac{1}{1-\pi_n}\right]^{n-1}$$

For each value of x the function $F(x)^{n-1}$ is decreasing in n. Therefore, the point π_n at which the above condition holds is strictly increasing in n (since $\hat{q}_n$ can never equal q_H in equilibrium). This fact plus inspection of the agent's optimization problem reveals that $\hat{q}_n$ is also increasing. ■

Conceptually, each agent contributes only a small amount to the probabilistic chance of development when the number of developers is not too small. Since the previous Lemma shows that the chance of the other agents developing is increasing, it stands to reason that the overall development probability can not go down by very much when the size of the developer base is not too small. The following theorem makes this precise without relying on any particular distributional assumption.

Theorem 1 *If the population of user-developers is n, then any decline in the development probability resulting from adding one more user-developer is less (in magnitude) than $\frac{1}{(n-1)e}$.*

[12] An example in which the development probability always falls with the population size is when the value-to-cost distribution function is given by $F(q) = q^2/4$.

Proof: Letting p_n denote the probability that any given agent develops in equilibrium when there are a total of n developers, the change in the development probability is

$$(1-\gamma_{n+1})-(1-\gamma_n)=\gamma_n-\gamma_{n+1}=(1-p_n)(1-\pi_n)-(1-p_{n+1})(1-\pi_{n+1})$$
$$\geq (1-p_n)(1-\pi_n)-(1-p_{n+1})(1-\pi_n)=(1-\pi_n)(p_{n+1}-p_n)\geq -p_n(1-\pi_n)$$
$$=-p_n(1-p_n)^{n-1}\geq \min_{p\in[0,1]}\left[-p(1-p)^{n-1}\right]=-\frac{1}{n}\left(\frac{n-1}{n}\right)^{n-1}>\frac{-1}{(n-1)e}$$

where the last inequality follows from the fact that $\left(\frac{n-1}{n}\right)^n$ converges monotonically to e^{-1} from below. ■

This bound on the possible decrease in the development probability converges rapidly to zero. This suggests that for large projects there is little chance that growth in the developer base will lead to fewer developments.

The previous Lemma also implies that each agent is better off in expectation when the population increases. The reason is that the probability that another agent develops the project increases with n, which means each individual is better off (in expectation) unconditional on any realization of his or her own cost and value. In equilibrium, higher values of n deliver higher option values of doing nothing, without lowering the return on innovation.

Insofar as social welfare can be expressed as the sum of individual welfare, society is better off in expectation as well. It is not true, however, that individuals or society are better off in each state of nature.[13]

Theorem 2 *Expected social welfare is increasing in n. Moreover, the expected welfare of each user is increasing in n.*

Proof: Denote the expected payoff to agent i, conditional on his or her type and the total number of agents n, by $x_i(v_i, c_i, n)$. Then

$$x_i(v_i,c_i,n)=\max\left[v_i\pi_n,\ v_i-c_i\right]\leq\max\left[v_i\pi_{n+1},\ v_i-c_i\right]=x_i(v_i,c_i,n+1)$$

since $\pi_n\leq\pi_{n+1}$.

[13]For example, there are states in which the addition of another user results in the project not being developed when it would have been developed in the absence of the marginal user. This is true precisely because the threshold $\hat{q}_n$ is rising with n. However, individuals are better off in states of the world where the marginal user develops and they themselves do not, but would have otherwise.

Hence agent i's payoff is increasing in n conditional on his or her type. But since this is true for every type of i, his or her ex-ante payoffs Ex_i are also increasing in n. Finally, observe that agent's payoffs are never negative.

It follows that expected social welfare with n agents can be expressed as

$$\sum_{i=1}^{n} Ex_i(v_i, c_i, n) \leq \sum_{i=1}^{n} Ex_i(v_i, c_i, n+1) \leq \sum_{i=1}^{n+1} Ex_i(v_i, c_i, n+1)$$

where the last term is expected social welfare with $n+1$ agents. This proves the theorem. ■

3.1 Limiting Results

One of the major arguments for why the open source paradigm should be successful is that open source code permits an extremely large labor force (potentially the entire Internet community of programmers) to bring its skill and insight to bear on a problem.[14] The notion that the open source method can marshal considerable intellectual power seems to be taken seriously by some major firms. Consider the following excerpt from an internal Microsoft document, which assesses the threat of open source software (or OSS as it is referred to below):[15]

> "The ability of the OSS process to collect and harness the collective IQ of thousands of individuals across the Internet is simply amazing....Linux and other OSS advocates are making a progressively more credible argument that OSS software is at least as robust– if not more– than commercial alternatives."[16]

It is thus natural to explore the behavior of the model as the pool of user-developers grows large. The limiting probability of innovation is investigated

[14] In the case of bug fixing, this notion is captured by Linus' Law, which states, "Given enough eyeballs, all bugs are shallow."

[15] These so-called 'Halloween Documents' can be read at `opensource.org`. Their authenticity has been confirmed by Microsoft itself at `www.microsoft.com/ntserver/nts/news/mwarv/linuxresp.asp`.

[16] As a practical matter, it is not clear exactly how successful the open source paradigm is either in absolute terms or relative to proprietary alternatives. Much evidence is merely anecdotal. However, Miller, Koski, Lee, Maganty, Murthy, Natarajan, and Steidl (1998) found that failure rates of commercial versions of UNIX utilities ranged from 15-43% in contrast to failure rates of 9% for Linux utilities and just 6% for GNU utilities (which are also open source).

first. Then, the issue of the distribution of costs and redundant effort is considered.

It is important to note that another major argument for why the open source system should be successful is that it engenders interactions and synergies between developers. These are explicitly ruled out in this model.

3.1.1 Development Probability

Consider what happens to the probability of development $1-\gamma_n$ and the probability π_n that any $n-1$ of the n users develops the software when the population grows large. The following is immediate.

Theorem 3 *Both π_n and γ_n have limiting values π^* and γ^*, respectively. In particular*

$$\gamma^* = \lim_{n\to\infty} \gamma_n = \frac{1}{q_H}$$
$$1-\pi^* = \lim_{n\to\infty} (1-\pi_n) = \frac{1}{q_H}$$

Proof: In fact, each of the above limits implies the other, since, by definition, $1-\pi = \gamma^{\frac{n-1}{n}}$. Hence, only the first limit will be derived. It has already been shown that a unique symmetric equilibrium exists for each n. Hence, all that needs to be demonstrated is that for any $\epsilon > 0$ there exists an N such that for $n > N$ the equilibrium value of γ_n lies in $(\gamma^* - \epsilon, \gamma^* + \epsilon)$.

Let $\epsilon > 0$ be given. It is clear that $1/(\gamma^* + \epsilon) < q_H$ and hence for some N_1 it is the case that $n > N_1$ implies $(\gamma^* + \epsilon)^{(1-n)/n} < q_H - \eta_1$ for some $\eta_1 > 0$.

Since $F(q_H) = 1$ and F is strictly increasing on its support, it must be that for $n > N_1$

$$F\left[(\gamma^* + \epsilon)^{\frac{1-n}{n}}\right] < 1 - \eta_2$$

for some $\eta_2 > 0$, which implies that

$$F\left[(\gamma^* + \epsilon)^{\frac{1-n}{n}}\right]^n$$

converges to zero. In particular, there is an N_2 such that (2) can not be satisfied at $\gamma^* + \epsilon$, or for any greater value, when $n > \max[N_1, N_2]$. This is so because F is an increasing function and because the map $x \mapsto x^{(1-n)/n}$ is decreasing.

Now consider $\gamma^* - \epsilon$. There is some value N_3 such that $n > N_3$ implies that $(\gamma^* - \epsilon)^{(1-n)/n} > q_H$ and hence that $F\left[(\gamma^* - \epsilon)^{(1-n)/n}\right]^n = F(q_H) = 1$

since F is a distribution function. This implies neither $\gamma^* - \epsilon$ nor any point less than it can satisfy (2).

One can now conclude that for $n > \max[N_1, N_2, N_3]$ it must be the case that $\gamma_n \in (\gamma^* - \epsilon, \gamma^* + \epsilon)$. Since ϵ was arbitrary, the result follows. ■

This is intuitive because, in the limit, only the agents with the highest value-to-cost ratios will develop the software. Hence, the asymptotic probability of no development must be such that it keeps an agent of type q_H indifferent. This type of result also appears in previous works on the public provision of private goods, e.g. Chamberlin (1974).

This result is robust to many modifications of the model. For example, if people received slightly higher values when they wrote the program themselves or if the underlying distributions were different the conclusion that the limiting probability of development not equal one would still hold.

On the other hand, the bounded support of the distribution of value-to-cost ratios is important. If the value-to-cost distribution were unbounded, then development would take place with arbitrarily high probability as the population size grew large. To see that this must be so, suppose for the sake of contradiction that the limiting probability of development were less than one.[17] From an individual agent's viewpoint, this implies that the probability that one of the other agents develops is less than one. For an agent with a sufficiently high value-to-cost ratio, it is suboptimal to not develop independently. This implies that the probability that an individual agent develops does not converge to zero, a contradiction.

3.2 Costs and Redundancy

It has already been shown that even an infinite number of "eyeballs" might not lead to innovation. Here the potential for wasteful duplication of effort is considered. This issue is considered by Raymond (1998) in response to the assertion of Brooks (1995) that adding more programmers to most software projects only delays completion, resulting in unbounded waste in the limit.[18]

In agreement with Raymond (1998), it can be shown that redundant efforts and costs do not grow without bound in the present environment. To

[17] Technically, this argument should be made using the lim sup of the probability of no development. It follows then that the lim sup converges to zero, so that the limit exists and equals zero.

[18] This is a version of Brooks' Law. The idea is that many tasks can only be performed sequentially and that, task by task, more programmers need not hasten progress. The model presented in the present paper clearly is not sequential and thus cannot be taken literally as a response to Brooks (1995).

this end, define p_n to be the probability that any individual in a population of size n chooses to develop. For a fixed population, the expected number of developments equals np_n.

Theorem 4 *The expected number of development efforts converges as the population grows. Precisely,*

$$\lim_{n\to\infty} np_n = \log(q_H)$$

Proof: It has already been shown that $\gamma_n = (1-p_n)^n$ converges to $1/q_H$ and so continuity of the natural logarithm implies that $n\log(1-p_n)$ converges to $-\log(q_H)$. A first-order Taylor expansion of the logarithm around 1 reveals that

$$n\log(1-p_n) = -n\frac{p_n}{1-\hat{p}_n}$$

for some $\hat{p}_n \in (0, p_n)$. Since p_n converges to zero, it follows that np_n converges to $\log(q_H)$. ■

The incentive to free ride is strong enough to bound the amount of redundant effort in the limit. Selfish agents willingly choose to restrict redundant effort. While perhaps a cynical conclusion, this theorem provides positive support to the open source paradigm. It is also true that total costs are bounded, and that in the limit only the least cost programmers develop.

Theorem 5 *The total expected costs of development borne by the open source community converge to* $c_L \log(q_H)$.

Proof: It is the case that $\hat{q}_n$ converges to q_H. This is clear since it has been previously demonstrated that the limiting probability of development is less than one, and since that result is incompatible with agents having non-negligible ex-ante development probabilities in the limit. Moreover, the limit of $\hat{q}_n$ must exist, since it is an increasing bounded sequence. Suppose for the sake of contradiction that it is not the case that only the agents with the lowest possible cost develop in the limit. Then, for some $\epsilon > 0$, the sequence $\{\bar{c}_n\}$ with elements defined by $\bar{c}_n = \sup\{c : v/c \geq \hat{q}_n\}$ contains an infinite number of elements that are greater than $c_L + \epsilon$. This implies that $\hat{q}_n < v_H/(c_L + \epsilon)$ an infinite number of times, which is incompatible with the fact that $\hat{q}_n$ converges to $q_H = v_H/c_L$. ■

This is in accordance with the perception in the open source community that it is those who find particular problems easy or interesting who end up

solving them. Of course, this theorem is a limiting result; in general, those who develop should not be expected to be those with the lowest costs.

It is important to note that this result relies heavily upon the rectangular support of $G(c, v)$. If the region of support were, for example, circular then it would not be true that the highest possible values of v/c corresponded to the lowest values of c.

Again, the assumption of bounded support for F is critical. If the support of F were unbounded, the amount of redundant effort would become unbounded as n grew. The reason is that users with extreme value-to-cost ratios would not find it optimal to tolerate even a tiny probability of no development, and hence would invest their own resources.

It is possible to say a bit more about the distribution of redundant efforts. Theorem 4 also implies that, regardless of the underlying joint distribution of values and costs, the (random) number of development efforts follows a well-defined distribution asymptotically.

Corollary 1 *The number of development efforts converges to a Poisson random variable with mean* $\log(q_H)$.

Proof: For each n, the number of development efforts can be expressed as the sum of n independent binomial variables with success probability p_n. We have $p_n \to 0$ and $np_n \to \log(q_H)$. It is known that the limiting distribution is therefore Poisson. See, for example, Ash (1972), page 348. ■

4 Comparing Open Source to Closed Source

In this section the relative performance of an open source system is compared to a closed source one and also to a constrained social planner's solution. To set a closed source benchmark, imagine that a software company has already sold a product to n individuals, but has not revealed the source code. There is a potential product enhancement that the firm can develop at cost c. The innovation has no internal consumption value to the firm. Assume that the firm will only produce if its expected revenue exceeds the opportunity cost c of having its engineers work on the program.

Now consider a social planner who wishes to maximize the expected sum of values less costs in the community. Assume that the social planner must assign each agent a rule to follow. Each agent's rule tells the agent whether to develop or not conditional only on his or her own private value and cost. These rules must be assigned prior to the determination of any randomness.

Thus, the social planner is constrained by the fact that all information is private.

Attention is also restricted to deterministic, symmetric rules. Given these restrictions, the planner instructs each agent to develop if and only if her value and cost pair (v, c) lie in some development region Δ. The following theorem describes this region.

Theorem 6 *If a social planner is constrained to offer each agent the same deterministic decision rule, then there are constants $a, b > 0$ such that each agent i is instructed to develop if and only if*

$$c_i \leq a + bv_i$$

Proof: This can be deduced by considering the action of an agent whose decision has no net impact on social welfare in expectation (given their valuation and cost). Consider a single person, say agent 1, on the boundary of Δ. If she develops, social welfare is

$$-c_1 + v_1 + E\sum_{i=2}^{n} v_i - (n-1)p_c c^* \tag{3}$$

where p_c is the probability that any other individual agent's value and cost pair lie in Δ, and c^* is the expected cost of that agent conditional on being in Δ. If this agent instead does not develop, welfare is given by

$$\left[1 - (1-p_c)^{n-1}\right]\left(v_1 + E\sum_{i=2}^{n} v_i^*\right) - (n-1)\,p_c c^* \tag{4}$$

Where v_i^* is the valuation of agent i given that at least one of the last $n-1$ agents does in fact develop. Of course, the values p_c, c^* and v_i^* are endogenous, in that they depend upon the rule that has been assigned. This does not influence the present analysis.

Rewrite (3) in the following manner:

$$-c_1 + v_1 + \left[1 - (1-p_c)^{n-1}\right] E\sum_{i=2}^{n} v_i^* + (1-p_c)^{n-1} E\sum_{i=2}^{n} v_i^{**} - (n-1)p_c c^*$$

where v_i^{**} is the value of agent i given that none of the last $n-1$ agents develop. Since agent 1 is presumed to be on the boundary of Δ, social

welfare should be invariant in expectation to her decision. Equating (3) and (4) yields:

$$(1-p_c)^{n-1}\, v_1 + (1-p_c)^{n-1}\, E \sum_{i=2}^{n} v_i^{**} = c_1 \tag{5}$$

Letting $a = (1-p_c)^{n-1}\, E \sum_{i=2}^{n} v_i^{**}$ and $b = (1-p_c)^{n-1}$ completes the proof of this theorem. ■

One more result is easily obtained and will add to the discussion that follows. The open source scheme exhibits a lower development probability than that of the social planner.

Theorem 7 *When agents obey the socially optimal decision rules, the overall probability of innovation is higher than it is in the equilibrium of the open source regime.*

Proof: Let $\hat{\pi}_i$ denote the probability that some agent other than i will develop under the socially optimal scheme, and let π be the corresponding equilibrium probability under the open environment. Bearing in mind that the notation of Theorem 6 is such that $(1-p_c)^{n-1} = 1-\hat{\pi}$, observe that (5) implies that an agent of type (v, c) will be instructed to develop whenever

$$v - c \geq \hat{\pi} v - (1-\hat{\pi})\sigma$$

where $\sigma > 0$. This implies that any agent type who would develop in the open environment would also develop under the socially optimal scheme if it were the case that $\hat{\pi} \leq \pi$. In fact, strictly more types would develop so that, given the smoothness of the underlying distribution, it would follow that $\hat{\pi} > \pi$. This contradiction completes the proof. ■

Relative to the social optimum, the level of development is too low in the open environment. Furthermore, the distribution of effort is inefficient in the sense that some types that develop under the open regime might not develop under the social planner's solution. These will be types with high values and high costs. Facing free riding (and the lower probability that someone else will develop) in the open regime compels these agents to innovate themselves. The social planner, however, will not want very high cost agents to develop. Such agents are compensated, so to speak, by the fact that the planner instills a regime in which the probability that other agents develop is higher than in the open regime.

A diagram is useful in comparing the three possible systems. Suppose that valuations are measured on the horizontal axis and costs on the vertical axis. The decision rules that would be followed by individuals under the three systems can be shown graphically. Each development region is the area underneath a particular ray in the value-cost space. The monopoly rule is a horizontal ray since the firm develops whenever its cost is low relative to the expected profitability of the project, which is constant (because it has no internal consumption value for the project). The open source rule is a ray emanating from the origin at a slope less than one, and the optimal scheme a ray emanating from a point above the origin at a slope less than that of the open rule.

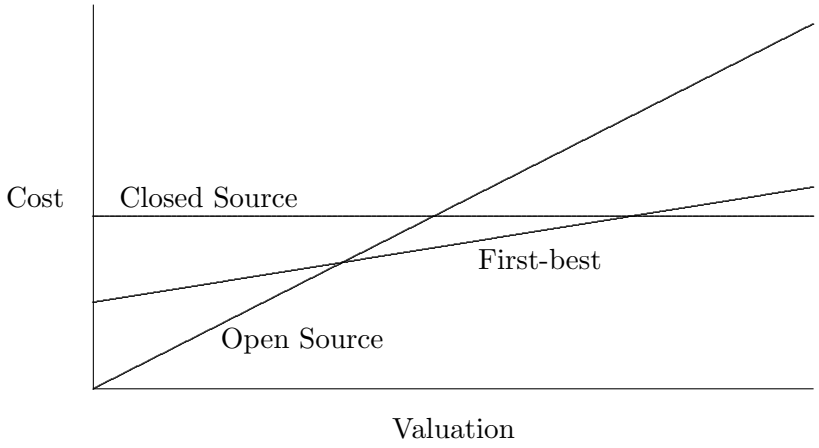

Figure 1: Comparison of the Three Systems

Some heuristic comments are in order. While it might appear to be a negative that a profit-driven firm only cares about the monopoly profits it can extract, the fact that the firm cares about the valuations of the other consumers at all speaks well of the firm. On the other hand, the resources of the firm are limited in that it can not access the entire talent pool of the Internet. This is an assumption of the model, but there are several reasons why it might be so in reality. First, when source code is closed, it is not even possible for individuals to know what their costs would be much less for the monopolist to know. This is certain to complicate contracting efforts. Also, as a practical matter, most open source programmers are already employed and choose to work on open source projects in their spare time. Their costs might include a random opportunity cost component that depends on their workload at their primary place of employment. Hence, even if it is clear

who the best engineer is for a given task, it might not be clear whether he or she will be available to perform the labor. Third, a firm might believe that revealing its source code on a wide basis might provide an edge to any competitors, present or future.

The open source system, in contrast, exploits the potential of all the users. This can (but need not) result in only low costs being borne.

Another plus for open code is that more information is being used. This is meant in the following sense. Each agent has access to his or her private information and could end up writing the software. The monopolist knows its value (i.e. expected revenue) and cost as well, but none of the individual users can exploit their own information when the source code is unavailable. More information is being "conditioned on" (although the information is not aggregated) when everyone has access to the code.

One might simply say that firms don't always know what people want, but people usually do. When source code is unavailable publicly, the human capital and insight present in the community as a whole cannot be harnessed.

A good example appears to be the Apache web server. This was developed from the original NCSA web server beginning around 1995. The people who developed Apache found that many changes to the NCSA were needed. Evidently, no firms were supplying these changes at prices that many webmasters were willing to pay. One might think that the dramatic changes taking place on the worldwide web were such that the webmasters had vastly superior information about their own needs. Arguably, the open nature of Apache allowed important developments to occur more rapidly than would have otherwise been possible.

5 Other Open Source Issues

5.1 An Empirical Puzzle

A puzzle in the open source community is why some obviously useful software does not get written (or is not fully developed). For example, while open source word processors and spreadsheets do exist, it is fair to say that only recently have they begun to be comparable in quality to, for example, Microsoft Office.[19] On the other hand, hundreds of other free utilities and applications exist. In this section it is argued that a natural correlation between the human capital and the production technology of workers will

[19]For example, the home pages of open projects like Gnumeric and KOffice admit that a lot more development is needed. This again highlights a limitation of a static model with a single project rather than a dynamic one with varying degrees of progress.

tend to lead to the production of certain types of programs (like computer utilities and Internet protocols) but not others (like word processors and spreadsheets).

An argument put forth by Eric S. Raymond[20] is that open source programmers wish to establish a reputation for ingenuity in the greater hacker community (Raymond 1998). Thus, projects that are considered more exciting are more likely to be developed.

The model developed here admits a simple alternative explanation. Consider two possible applications, an enhancement of a word processor and an addition to a networking utility. Inasmuch as people who are most likely to value the networking utility are also most able to write the addition, a natural negative correlation exists between value and cost. Not surprisingly, such negative correlation can easily lead to heightened levels of development.

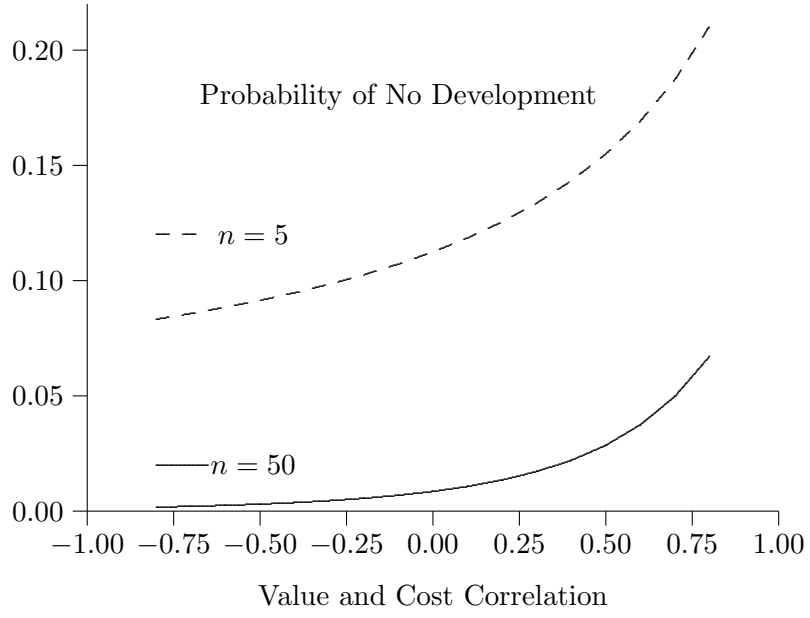

Figure 2: Correlation Diagram

In Figure 2 the situation is considered when value and cost have a joint log-normal distribution with correlation coefficient given by ρ, and two population sizes. Changing the correlation coefficient does not alter either marginal distribution, but influences the distribution of the value-to-cost

[20]Eric S. Raymond is a programmer and well known open source software advocate. He was influential in Netscape's 1998 decision to release its browser source code.

ratio. As the correlation between value and cost falls, the open source community performs better, as measured by the increase in the development probability.

Another consideration is that programs geared towards non-specialists may require extensive and costly usability testing. If those at great ease with software are writing programs for themselves, it is not surprising that they might neglect to include certain "friendly" features.

5.2 Modularity and Incremental Development

Many open source projects receive code contributions that are individually quite small. As a whole, the sum of these contributions might be quite valuable. Prevalent among open source proponents is the notion that one reason the open environment can be successful is that there are many small tasks that can be completed for any particular project. With many small tasks, the argument runs, it becomes more likely that any individual will find it worthwhile to contribute, increasing aggregate development.

Here the issue of whether scope for incremental innovation should lead to heightened development is investigated by extending the basic model. Two different open environments are considered, one of which is "modular" and the other of which is "nonmodular." As will be clear, the modular environment admits incremental innovation while the nonmodular environment does not.

Suppose that there are k tasks or projects. Each user-developer receives an independent draw from $G(c, v)$ for each of these projects. Define the modular environment to be one which is a k-replica of the standard model. That is, individuals simultaneously decide which if any of the projects to complete. As long as at least one agent chooses to develop a particular project, all agents receive their valuations for that project.

Define the nonmodular environment to be one in which agents receive their valuations for a project only if at least one user has individually completed all k tasks. That is, the entire collection of projects is worthless unless a single developer has completed each of them by herself. Hence, each agent must decide *ex-ante* whether to develop all or none of the projects. It is straightforward to show that this implies that the decision rule of each agent is to develop all k projects if and only if

$$z_i^k = \frac{\sum_{j=1}^{k} v_i^j}{\sum_{j=1}^{k} c_i^j}$$

is sufficiently large. It turns out to be the case that whether a modular environment will generate more development in expectation depends critically on the size of the developer base.

Theorem 8 *Define N^* as follows:*

$$N^* = 1 + \frac{\log\left(\frac{Ec}{Ev}\right)}{\log\left(F\left(\frac{Ev}{Ec}\right)\right)}$$

For any fixed $n > N^*$ there exists some K such that for all $k > K$ the expected number of tasks completed in the modular case with n users and k possible tasks exceeds the number in the corresponding non-modular case. For any fixed $n < N^*$, there exists a K such that for $k > K$ the expected number of tasks completed in the modular case with n users and k possible tasks is less than the number in the corresponding non-modular case.

Proof: Let π^n_{mod} denote the probability that any of $n-1$ users develops any one given project in the modular case (this is independent of k), and let $\pi^{n,k}_{\text{nmod}}$ denote the probability that the development is made in the non-modular environment (that is, that all k tasks are undertaken together by at least one agent of $n-1$).

To prove the theorem, it need only be shown that when $n > N^*$ there exists a K such that for $k > K$ it is the case that $\pi^{n,k}_{\text{nmod}} < \pi^n_{\text{mod}}$, and that when $n < N^*$ there exists a K such that for $k > K$ it is the case that $\pi^{n,k}_{\text{nmod}} > \pi^n_{\text{mod}}$. For then elementary facts about expectations of sums of random variables will imply the theorem.

It will be shown that as k grows large the law of z^k_i places arbitrarily large probability on a neighborhood around Ev/Ec. Having shown this, it will follow that the nonmodular environment's limiting equilibrium, obtained by letting k grow large and holding n fixed, is identical for each n. This will in turn allow the two environments to be compared for a given population.

Observe that z^k_i can be expressed as

$$z^k_i = \frac{\frac{1}{k}\sum_{j=1}^{k} v^j_i}{\frac{1}{k}\sum_{j=1}^{k} c^j_i}$$

so that the law of large numbers implies that the corresponding law places arbitrarily high probability on any given neighborhood of Ev/Ec as k grows large.

The equilibrium of the nonmodular environment converges to one in which $\pi = 1 - Ec/Ev$. Since z_i^k is converging in probability to Ev/Ec for each user, any solution to the equation

$$\pi = 1 - F_k\left[\frac{1}{1-\pi}\right]^{n-1}$$

must be arbitrarily close to $1-Ec/Ev$, since the distribution function $F_k(z_i^k)$ is converging to a function that places an atom at Ev/Ec. Therefore, for fixed n, $\pi_{\text{nmod}}^{n,k}$ converges to $1 - Ec/Ev$ as k grows large.

Next, observe that $\pi_{\text{mod}}^n > \pi_{\text{nmod}}^{n,k}$ if and only if $n > N^*$. The proof is simple. If

$$1 - \pi > F\left[\frac{1}{1-\pi}\right]^{n-1}$$

then it must be that $\pi_{\text{mod}}^n > \pi$, whereas the opposite conclusion holds otherwise. Letting $\pi = 1 - Ec/Ev$, it is clear by directly solving for n that

$$n = 1 + \frac{\log\left(\frac{Ec}{Ev}\right)}{\log\left(F\left(\frac{Ev}{Ec}\right)\right)} = N^*$$

will exactly satisfy the above inequality. This proves the theorem. ■

When the number of potential developers is large enough the modular environment will outperform the non-modular one. It is better to work with a large number of upper tails that correspond to smaller projects than to work with a small number of averages that correspond to larger projects.

However, when the number of potential developers is small the modular environment does better in terms of development. The reason is that developers know that the project has no functionality unless all of the components are present. There may be parts of the whole that are high cost or low value to a given user. That user might nonetheless be willing to put "extra effort" in to be sure that the aggregate product, which she does value, will exist. Thus non-modularity will sometimes temper the free riding present in the open source development system.

Raymond (1998) asserts that good open source projects need to be developed initially by a small group and only later released to the general community for further improvement. Heuristically, developers need to have something sizable to "play with" before the open source model can be expected to do well.

5.3 The Completeness of Open Source Software

Some people are reluctant to experiment with open source software because there is an impression that such software tends to be less complete than corresponding closed source applications. It often seems that proprietary software is easier to learn, has more features, better documentation, and is more user friendly on the whole. In this section, the modular framework introduced above will be adapted to provide a theoretical explanation of this observation.

Imagine that there is not very much cost variation across projects, so that they are all of similar difficulty to the same programmer. Formally, suppose that the cost of development varies across developers, but not across projects holding the developer fixed. As the number of components k grows large, the chance that an open source project develops all possible projects is very small. On the other hand, if a profit-maximizing firm chooses to develop any of the projects, it will develop all of them.

Theorem 9 *In the modular environment, the probability that an open source community develops all k of the possible projects approaches zero as k grows large. However, a profit-maximizing firm that chooses to develop at all will develop each of the k possible projects.*

Proof: The probability that any one of the developments is made by the open source community is independent of k. Call this probability $(1-\gamma) < 1$. It is obvious that the probability of all developments occuring is $(1-\gamma)^k$ which clearly converges to zero as k becomes infinite.

On the other hand, a firm will choose to develop any particular component if the expected profitability exceeds costs. Under the maintained assumptions that all developments yield the same expected profits, and that the firm's costs don't vary across developments, it follows that it is profitable to develop all the projects if it is profitable to develop any one of them. ■

Admittedly, only the simplest demand functions are being considered. Nonetheless, the intuition seems solid: Firms care about expectations that could be highly similar across different, small features of a program. They are likely to develop many portions of a program if they develop any.

6 Conclusion

The open source software movement is not new. However, only with the striking success of Linux, coupled with the decisions of major firms such as

IBM and Sun Microsystems to open their source code has national attention been attracted.

It is striking that a paradigm for costly investment based upon the absence of property rights has produced such a wide variety of useful and reliable software. A simple model of open source software has been presented to facilitate understanding of the phenomenon, and to enable efficiency comparisons between it and the traditional, profit-driven method of development.

It has been shown that the superior ability of the open source method to access the Internet talent pool, and to utilize more private information, provides an advantage over the closed source method in some situations. Nonetheless, free riding implies that some valuable projects will not be produced, even when the community of developers becomes large. However, this same free riding also curbs the amount of redundant efforts in the limit.

Potential explanations for several stylized facts have been presented, including why some simple programs are not written while other very complex programs are, and why proprietary programs tend to be more complete than open source programs. Also, the advantage of the possibility of incremental development has been shown to depend on whether the developer base exceeds a critical mass or not; this provides a theoretical explanation for why open source is a good development model when a base product has already been completed but not a good means of producing the base product itself.

The open source movement is gaining attention. Many questions concerning the movement remain unanswered. In this paper the seemingly prior question of how well an open source community will function given that it exists has been addressed. The answers provided hopefully will aid in the investigation of other aspects of open source software.

References

Ash, R. B. (1972): *Real Analysis and Probability.* Academic Press, New York.

Bergstrom, T., L. Blume, and H. Varian (1986): "On the Private Provision of Public Goods," *Journal of Public Economics*, 29.

Bliss, C., and B. Nalebuff (1984): "Dragon-slaying and Ballroom Dancing: The Private Supply of a Public Good," *Journal of Public Economics*, 25.

Brooks, F. P. (1995): *The Mythical Man-Month: Essays on Software Engineering*. Adison Wesley, Reading, MA.

Chamberlin, J. (1974): "Provision of Collective Goods as a Function of Group Size," *American Political Science Review*, 68.

Dixit, A., and C. Shapiro (1986): "Entry Dynamics with Mixed Strategies," in *The Economics of Strategic Planning: Essays in Honor of Joel Dean*, ed. by L. Thomas, III. Lexington Books, Lexington, MA.

Hecker, F. (1998): "Setting Up Shop: The Business of Open-Source Software," http://people.netscape.com/hecker/setting-up-shop.html.

Lerner, J., and J. Tirole (2000): "The Simple Economics of Open Source Software," *NBER Working Paper 7600*.

Miller, B. P., L. Fredrikson, and B. So (1990): "An Empirical Study of the Reliability of Unix Utilities," *Communications of the ACM*, 33.

Miller, B. P., D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl (1998): "Fuzz Revisited: A Re-examination of the Reliability of Unix Utilities and Services," University of Wisconsin Computer Science Working Paper.

Palfrey, T. R., and H. Rosenthal (1984): "Participation and the Provision of Discrete Public Goods: A Strategic Analysis," *Journal of Public Economics*, 24.

Raymond, E. S. (1998): "The Cathedral and the Bazaar," http://www.tuxedo.org/∼esr/writings/cathedral-bazaar/cathedral-bazaar.html.

Stallman, R. M. (1996): *GNU Emacs Manual, version 19.33*. Free Software Foundation, Boston, MA.