December 2002

Quality and Profits Under Open Source Versus Closed Source

Birendra Mishra
University of Texas at Dallas

Ashutosh Prasad
University of Texas at Dallas

Srinivasan Raghunathan
University of Texas at Dallas

QUALITY AND PROFITS UNDER OPEN SOURCE VERSUS CLOSED SOURCE

Birendra Mishra
School of Management
University of Texas at Dallas
Richardson, TX USA
bmishra@utdallas.edu

Ashutosh Prasad
School of Management
University of Texas at Dallas
Richardson, TX USA
aprasad@utdallas.edu

Srinivasan Raghunathan
School of Management
University of Texas at Dallas
Richardson, TX USA
sraghu@utdallas.edu

Abstract

The open source model of software development has received substantial attention in the industry and popular media; nevertheless, critics frequently contend that open source software are inferior in quality compared to closed source software because of lack of incentives and project management, while proponents argue the opposite. This paper examines this quality debate by modeling and analyzing software development under open and closed source environments. The results show no dominant quality advantage of one method over another under all circumstances. Conditions under which each method can generate higher quality software are examined.

Keywords: Open source, software quality

1 INTRODUCTION

Recent years have seen an increasing interest in the open source movement as a new paradigm for software development. Open source refers to the use of shared source code, open standards, and collaboration among software developers and users worldwide to build software, identify and correct errors, and make enhancements (O'Reilly 1999). Unlike the traditional (proprietary) paradigm of software development, users have free access to the source code, which they can modify to correct software bugs, port the software to new hardware or software platforms, solve additional problems, create add-on software programs, or simply use it for free. Enhancements submitted by individual users are often fed back to the original source code for public use. Table 1 provides short descriptions of some commonly used open-source software that are widely used by companies including some of today's most powerful e-commerce Websites like Yahoo, Cisco Systems, and C-Net.

Championed by the philosophy that software is a public good and should be freely shared, used, and codeveloped by all, the open source movement challenges the traditional paradigm of software as a proprietary good, to be guarded carefully via copyrights or patents and licensed or sold to users for profit. The traditional paradigm (referred to as *closed source*) treats software development as a highly specialized process, managed best by a localized team of highly qualified developers, careful project management, and occasional enhancements in the form of new releases. In contrast, open source software is based on the principles of continuous improvement (implemented via frequent releases), collaboration among developers and users (irrespective of geographical locations or employing firms), and adherence to open standards (implemented via open-source licenses). As described by Raymond (1999), open source represents a "bazaar style" of software development, in contrast to the "cathedral style" emphasized by closed source software development.

Table 1. Examples of Open-Source Software

Software	Description	Closed-Source Competitors
Operating system:		
Linux	Linux is the predominant enterprise server operating system in use today with over 7 million users.	Windows NT (Microsoft), OS/2 (IBM), HP-UX (HP)
FreeBSD	FreeBSD, Open BSD, and other Berkeley UNIX derivatives boast an estimated 1 million users.	Solaris (Sun), HP-UX (HP)
Programming languages:		
Perl	Larry Wall's Perl language is the engine behind most "live content" on the Internet, and is used by over 1 million users.	ASP (Microsoft)
Python	Guido van Rossum's Python language is used by over 325,000 users.	
Programming tools:		
GNU project	A high-quality set of programming tools, including gcc C , g++ C++ compiler, emacs editor, and gdb debugger from the Free Software Foundation's GNU project.	Visual C++ (Microsoft), Cold Fusion (Allaire)
Web server:		
Apache	Apache is currently deployed in over 53 percent (1 million) of today's web servers, well ahead of second-placed Microsoft's IIS at 23 percent.	Internet Information Server (Microsoft)
Web client:		
Mozilla	Netscape's next-generation web browser Mozilla, is the open-source version of its popular browser product Communicator.	Internet Explorer (Microsoft)
Internet infrastructure:		
BIND	Berkeley Internet Name Daemon servers are the core of the Internet's Domain Name Service (DNS) system.	
Sendmail	Sendmail is the primary e-mail transport/forwarding agent on the Internet. Used by 80 percent of all Internet sites.	
Miscellaneous:		
Ghostscript	Postscript editor and printer	Adobe Acrobat
StarOffice	Sun Microsystems' open source word processor, spreadsheet and other general-purpose office application software.	Microsoft Office

Critics contend that the quality of open source software suffers from the "free rider" problem. However, proponents claim that software contributors are not necessarily altruistic; programmers contribute to source code for social recognition and prestige in the open source community and to signal their talent to prospective employers, which may lead to job, consulting, or other career opportunities (Lerner and Tirole 2000).

In addition, critics suggest that the lack of formal project management in bazaar style software development undermines the product's quality. However, open source proponents argue that the quality of open source software stems not from its management but from its openness. Free access to the source code ensures that the code is tested and retested by a worldwide user base, leading to timely identification of bugs and opportunities for software enhancements, hence, is more reliable. In contrast, closed source software has a lower likelihood of error identification and correction, but closed source firms have a strong incentive to continually improve their program to ensure that clients buy the updates later. Limited coordination in an open source environment may lead to some duplication of effort between programmers trying to correct the same error and, hence, it is unclear whether having multiple users spend resources on improving a common software necessarily improves its quality over having a closed firm with coordinated efforts and formal project management.

The proponents of both paradigms point to the inherent incentives built into respective approaches to build quality software, with respect to software pricing and the approach to software development (e.g., cathedral style versus bazaar style). This paper focuses on the impact of inherent incentive structure of the closed and open source approaches on the software quality using *game theoretic* analysis to examine whether quality may improve or suffer in an open source model, compared to that in a closed source model. Our findings indicate that despite the free nature of open source software, built-in incentives for software developers in the open source model can enhance software quality to be comparable to or exceed closed source quality.

The remainder of the paper proceeds as follows. In the next section, we discuss the quality debate surrounding software development, specifically focusing on open source software. Section 3 discusses the modeling framework and assumptions. Section 4 examines how quality improvement comes about in a monopoly market. Section 5 examines quality improvement when closed source software is in monopoly and compares the result to those in the preceding section. The final section concludes with discussion and directions for future research.

2 THE QUALITY DEBATE IN OPEN SOURCE SOFTWARE

High quality software is vital for most firms because software applications are rapidly emerging as a firm's central nervous system; not only do they control a firm's everyday operations, but they also enable or constrain the ability to make and implement strategic business decisions (Prahalad and Krishnan 1999). The traditional conception of software quality is centered on a product-centric, conformance view of quality (Prahalad and Krishnan 1999). Benchmarking software based on predefined specifications assumes that it is possible to specify *ex ante* the entire range of features expected of a software. However, in reality, different users have different expectations of the same software and users' expectations of software evolve with time. For instance, some users may view performance and reliability as the key features of software, while others may consider ease of installation and maintenance as key features of the same software. Therefore, software applications today must do more than just meet technical specifications; they must be flexible enough to meet the varying needs of a diverse user base and provide reasonable expectations of future enhancements. Changing user perceptions necessitate a new way of conceptualizing and assessing software quality by synthesizing the conformance, service, and innovation aspects of the software and its vendor (Prahalad and Krishnan 1999). Conformance refers to a software vendor's ability to deliver the right product; service refers to its ability to customize the product to specific user needs; and innovation refers to its ability to deliver a continuous stream of novel features in the form of future upgrades. Although quality can be multidimensional, for the purpose of this analysis we distill all three dimensions into a single dimension.

Comparing open source with closed source software based on the above conceptualization of quality suggests some interesting implications. First, open access to source code allows greater opportunities for customizing open source software to the specific needs of individual users than closed source software. In fact, such customization is encouraged and widely practiced by the open source community, while closed source software that provide clients with access to source code (e.g., SAP R/3) proclaim that any modification to source code will result in loss of customer support, thus limiting the product's customizability to a predefined set of options. Second, continuous innovation and upgrades require soliciting suggestions and feedback from software users. While open source programmers actively solicit inputs from users worldwide via listservs, Usenet groups, and bulletin boards, closed source software developers have limited interaction with a small group of users. Hence, any enhancement to closed source software will reflect the changing needs of a small user group and be smaller in magnitude and scope than that of open source software. In practice, upgrades to open source software are released much more frequently than upgrades to closed source software, and consequently, open source software such as Linux seem to have developed faster than comparable closed source software such as Windows NT. As examples, Linux has supported 64-bit processing since 1995 while NT became 64-bit ready in 2000; Linux supports the latest networking standards such as Ipv6 but NT does not; and Linux provides ports to more software development environments than does NT. However, open source projects do seem to lag their closed source counterparts in coordination of developers and project management, leading to some duplication of efforts by multiple developers, inefficient allocation of time and resources, and lack of attention to mundane software attributes such as ease of use, documentation, and support, all of which impact conformance to specifications.

Indeed, industry comparisons of open source versus closed source software are inconclusive or slightly in favor of open source. For instance, a comparative evaluation of Linux and Windows NT (respectively, open source and closed source server operating systems) by Bloor Research (1999) found that Linux is significantly superior to NT in three out of nine dimensions (availability, user satisfaction, and value for money), somewhat superior in three other dimensions (operational features, support, and scalability), comparable in two dimensions (interoperability and functionality) and inferior in one dimension (application availability). Some system administrators prefer Linux over Windows NT because of Linux's simpler hardware configuration requirements, lightweight nature, better multi-processing capabilities, and networking support; however, others prefer Windows NT because of better security, ease of management, and superiority in Java performance. The lack of a verdict on the open source versus closed source debate is reflected in the fact that closed source firms such as IBM, Sun Microsystems, and Netscape Communications, that profess value in the open source philosophy and routinely commit time, resources, and manpower to open source projects, are not yet ready to abandon their internal closed source projects.

When an individual programmer submits a proposed improvement to an open source software, a coordinating committee (consisting of experienced programmers) evaluates the merits of that proposal, examines possible side-effects of the suggested improvement, compares it with alternative ideas submitted by other programmers, and finally decides whether to incorporate the proposed improvements to the software, and if so, distributes the enhanced version as a new software release. As long as developers return some of their improvements to this committee, the software will continue to improve. Submission, evaluation, and acceptance of improvements can be viewed as a tournament, where one improvement is accepted from a set of submissions, and the programmer submitting that improvement gains visibility, respect, and future prospects within and outside the open source community (Lerner and Tirole 2000).

Finally, as suggested by the open source examples listed in Table 1, the open source model appears most viable for generic context-independent applications such as operating systems, network software, word processors, and spreadsheets, where no specialized user inputs are required. Our domain of study for open source is likewise restricted to general purpose software used by a large number of end users who are price and quality takers (as opposed to a specific use software contract requiring exact price and quality specifications). The model is thus not applicable for highly specialized business applications such as customized procurement systems or payroll systems, where programmers cannot code without knowing the users' requirements and the quality of the software product is determined by its meeting contractual specifications.

We summarize this section with a table listing the key features and relevant differences between open source and closed source software. These features provide the conceptual underpinnings for modeling open source and closed source software development within an economic framework.

Table 2. Conceptual Underpinnings of Modeling Framework

Open Source	Closed Source
1. Programmers freely provide improvements for recognition if their improvement is accepted.	1. Programmers are hired by a software firm and work for compensation.
2. If the program is popular, a very large number of programmers may work on it.	2. The number of programmers working on the code is limited by the firm's resources.
3. A coordinating committee or an open forum for exchange selects the best software quality improvements.	3. Coordination is imposed by the firm to reduce overlap of efforts.
4. Programmers compete with each other for recognition (tournament).	4. Programmers do not compete with each other.
5. Users do not pay a price; hence particularly attractive to users.	5. Users pay a price to offset the costs of research and development that the software firm invests.
6. Less time and effort is spent by individual user/programmer on research and development, but there are more programmers.	6. More time and effort spent on research and development by software firm, but there are fewer programmers.

3 THE MODELING FRAMEWORK AND ASSUMPTION

In this section, we present a software development model from both the consumer and developer perspectives. Table 3 provides a detailed summary of notations used in this paper. On the consumer side, we assume that the consumer valuation V of a software with quality Q is given by:

$$V = tQ^{\theta} \tag{1}$$

Table 3. List of Main Notation

Functions, Parameters, and Strategy Elements	Notation
Functions and Parameters	
Value to consumers	$V = tQ^{\theta}$
Quality of the software (open source, closed source)	$Q(Q_{os}, Q_{cs})$
Individual developers quality	$Q_i = q_i + \varepsilon_i$
Distribution of Q_i	$F_{Q_i}(Q) = q_i + \varepsilon_i$
Coordination parameter	α
Stochastic noise component	ε_i
Cost of quality	$C(q_i) = c\lambda_i q_i^2 / 2$
Index for cost of high and low type	$i \in (L, H)$
Proportion of high type programmers	β
Utility of a programmer	$w(y) = y^{v}$
Risk aversion parameter	v
Number of open source programmers	N_{os}
Market Size	D
Demand for software of quality Q at price P	$\Delta(Q, P)$
Reward for the winning open source programmer	$R(D)$
Variable cost to the firm for hiring M_{cs} employees	$C(M_{cs}) = kM_{cs}^2$
Strategy Elements	
Quality choice by the individual developer	q_i
Number of closed source programmers	M_{cs}
Price of the closed source software	P
Fraction of total profit retained by the firm	ρ

Consumers are heterogeneous in their valuations. We model this heterogeneity by assuming that t is distributed uniformly between 0 and 1 without loss of generality. The concave utility function of these consumers is captured by the parameter $\theta < 1$. There are a total of D consumers. Thus, in a monopoly, the demand for a software with quality Q offered at a price P, $\Delta(Q, P)$ is given by

$$\Delta(Q, P) = D(1 - P/Q^{\theta}) \tag{2}$$

On the developer side, the software is developed by a set of programmers. For individual programmers, quality is generated by the time and effort spent on the project. Programmer i chooses an average quality output q_i by applying the requisite effort. Some improvements can also come randomly, in a serendipitous manner, or by "stroke of luck." Additionally, some randomness is inherent due to the subjective nature of the evaluation process. Therefore, the evaluated quality Q_i is modeled as a random variable with two parts: a deterministic component (function of programmer effort) and a random term:

$$Q_i(q_i) = q_i + \varepsilon_i \tag{3}$$

The cumulative distribution function (c.d.f.) of Q_i is denoted $F_{Q_i}(Q)$.

Programmers' costs of quality $C(q_i)$ is assumed to be increasing and convex, i.e., $C'(q_i) > 0$ and $C''(q_i) > 0$. We know that programmers differ in their skill and experience. Presumably, all types of programmers can ultimately get a job done but more skilled and experienced programmers will find it less costly in terms of time and effort to do so. Thus, one may differentiate programmer types by their cost of programming parameter (Lazear and Rosen 1981). For simplification of analysis, we assume that programmers belong to one of two types: high (H) and low (L); however, this simplification is not critical to our results. For the same level of output quality, a high type programmer incurs a lower cost compared to a low type. More specifically, we assume that the programmer's cost of quality is given by the cost specification

$$C(q_i) = \frac{c\lambda_i q_i^2}{2}, \; i \in \{L, H\} \tag{4}$$

where, c is a cost parameter, and λ_i ($\lambda_H < \lambda_L$) indexes the relative cost between the high and low type programmers. Note that this cost specification satisfies the earlier stated requirements of the cost function: $C'(q_i) > 0$ and $C''(q_i) > 0$. The parameter c is likely to have a value less than 1 in open source models, and is normalized to 1 in closed source models. The value $c < 1$ in open source implies that open source programming may be viewed by some programmers as a hobby or may be done for altruistic motives (Lerner and Tirole 2000). We denote the number of programmers in high and low types by $N_H = \beta N$ and $N_L = (1 - \beta)N$ respectively, where N is the total number of programmers working on the software.

The quality choice of individual programmers determines total quality. We denote the *effective* quality that results in software improvement as $\alpha \sum q_i$ where a is the coordination parameter. The coordination parameter a is normalized to 1 in the case of open source software, which has little, if any, coordination among programmers. In the case of closed source, α is likely to be much greater than 1 because the closed source approach is designed to achieve a synergistic outcome that they could not have achieved individually. The utility specification for a programmer is given by

$$w(y) = y^{\nu} \tag{5}$$

where $\nu \leq 1$ is the risk aversion parameter. This power function is fairly standard for capturing risk aversion and is widely used in economics and business literature.

The above model applies to both closed source and open source software. The difference between closed source and open source approaches lies in the manner in which the programmers are rewarded for their efforts. In the case of open source, the programmers compete for recognition by participating in a tournament. In the case of closed source, the firm shares its profit with its programmers.

4 QUALITY UNDER OPEN SOURCE

We assume that the open source environment consists of a set of N_{os} programmers who submit improvements to the open source code, where $N_{os} \geq 2$. There is a coordination committee that costlessly evaluates submissions as a tournament and selects a winner. The role of the coordination committee may be formal or informal. For instance, the winner may be selected informally through mutual consensus among open source users, while a rigid hierarchical structure may govern which improvements, or variations of an improvement, will be incorporated into the software, as in the case of Linux. Let D_{os} be the number of open source users. Since the software is and free, the entire market should adopt the software regardless of its final quality. Thus, D_{os} is equal to the size of the market D in this monopoly case.

The winning programmer obtains a reward valued at $R(D)$ and there is no reward for the rest of the field. The value of the reward is expected to increase with the size of the user base, i.e., $R'(D) > 0$. This reward may include social recognition and prestige in the open source community, job offers from firms, consulting opportunities, selection to the board of directors of open source firms, and so forth (Lerner and Tirole 2000). Lerner and Tirole propose that open source programmers have two incentives to participate: the *ego gratification incentive* and the *career concern incentive*. Whereas the latter is a pure monetary concern, the ego gratification incentive is a function of the number of individuals who use the open source software. Reward is a function of the total number of users of the open source software, $R(D)$ can be interpreted as pertaining to both intrinsic and extrinsic rewards.

Open source programmers work independently of one another, because they are geographically dispersed and because of the incentive structure that forces them to compete with one another for the reward. In contrast, closed source software development encourages collaboration between programmers.

An individual programmer enters the tournament with the following expected payoff:

$$U_i = p(q_i, q_{-i})R(D)^{\upsilon} - C_{OS}(q_i) \tag{6}$$

where, $p(\cdot)$ is the probability of the programmer winning the tournament and $q_{\text{-i}}$ is a vector of the qualities of the remaining submissions. The probability of the programmer winning the tournament is the probability that his entry is evaluated better than the remaining $N_{\text{os}} - 1$ entries, i.e.,

$$p(q_i, q_{-i}) = prob[Q_i \geq \max\{Q_j\}_{j \neq i}] \tag{7}$$

Evaluating this probability requires the distribution of $\max\{Q_j\}_{j \neq i}$. Substituting equation (7) into the programmer's utility function and obtaining the quality choice of the programmer and the expected improvement, we reach the following proposition:

Proposition 1: *The optimal quality level of each programmer,* q_{os}, *is obtained as the solution of the series of* N_{os} *equations:* $\frac{\partial p(q_i, q_{-i})}{\partial q_i} R(D)^{\upsilon} - \frac{\partial C_{OS}(q_i)}{\partial q_i} = 0$, $i \in [1, N_{OS}]$. *The expected quality of the winning software is* $\int_0^{\infty} [1 - F_{Y_N}] - \int_{-\infty}^{0} F_{Y_N}$ *where* $F_{Y_N} = \prod_{j=1}^{N_{OS}} F_{Qj}$.

(Proofs are in the Technical Appendix)

Proposition 1 provides a general characterization of software quality in the open source model. However, more specific insights require imposing more structure to this proposition. For this purpose, we make the following assumption.

(Tournament Winner Assumption) The probability of programmer i winning the tournament is given by the function:

$$p(q_i, q_{-i}) = \frac{q_i}{\sum_{j=1}^{N_{OS}} q_j} \tag{8}$$

This probability can be derived both from Proposition 1 with a specification of the random variable based on the Gumbel distribution. The above assumption has been used widely in economics literature including the tournament literature (Rosen 1986). Although high cost programmers have a low probability of winning the reward, they do not voluntarily drop out. The first order condition is:

$$\frac{R(D)^{\upsilon}[\sum_{j=1}^{N_{OS}} q_j - q_i]}{[\sum_{j=1}^{N_{OS}} q_j]^2} - c\lambda_i q_i = 0 \tag{9}$$

The solution is bounded away from zero for all programmer types implying participation from *both* high and low types. Equations for the low and high cost programmers can be written and solved for the following results.

Corollary 1.1: The ratio of high and low productivity programmers' quality levels are $\frac{q_H}{q_L} = \frac{\lambda_L}{\lambda_H}$. *Software quality is* $Q_{OS} = \sqrt{\frac{N_{OS} R(D)^v}{c} (\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L})}$.

The above result leads to the following interesting and valuable insights. First, from Corollary 1.1, the quality improvement Q_{os} is increasing in v, β, N_{os} , and D, and decreasing in c. The former shows that risk aversion leads to lower quality levels. Higher number of programmers and higher fraction of high productivity programmers also increases quality because of higher total effort. The improvements due to D follow from the logic that if reward from effort is increased, programmers are motivated to work harder.

Second, a particularly interesting result is that the quality of open source software is decreasing in c. Critics contend that open source software development is dependent upon people who consider it a hobby rather than a serious pursuit, and hence quality may suffer. In contrast, our model suggests that such hobby considerations decrease the programmer's cost of effort, and thereby increase the quality of open source software.

Third, the role of N_{os} in quality improvement is somewhat ambivalent. A larger number of programmers decreases the likelihood of any individual programmer obtaining the reward and, therefore, acts as a disincentive to effort. At the same time, more programmers will provide a larger number of improvements and lead to higher quality software. Since the quality expression is increasing in the number of programmers, the latter is the dominant effect.

Fourth, some open source communities may require a critical quality threshold in addition to a programmer winning the tournament in order to realize any reward. For such a quality threshold Q^*, the expected reward is $p(Q_{os} \geq Q^*)p(q_i, q_{i-1})R(D)^v$ instead of $p(q_i, q_{i-1})R(D)^v$. $p(Q_{os} \geq Q^*)$ represents the proportion of projects undertaken under the open source model. Although analytically intractable, it can be seen that a quality threshold is detrimental to open source software quality since the reward is effectively reduced. If the quality threshold is sufficiently high, it is possible that the open source model may suffer from the "public good" problem because programmers unwilling or unable to meet the quality threshold may simply stop making contributions. When the threshold increases, the effective reward decreases, causing the individual programmer effort and the total quality to decrease. This, in turn, reduces the probability that the quality will be higher than the threshold, thus reducing the number of projects that will be taken up in the open source. This result may explain why relatively few new software projects are developed by open source.

Finally, although the above model examined programmers of two types (high and low ability), a related question is that whether open source quality can be improved by employing a more heterogeneous population of programmers. By setting $\lambda_N = 1 - \varepsilon$ and $\lambda_L = 1 + \varepsilon$ and examining the sensitivity of quality to heterogeneity, we see

$$sign[\frac{\partial Q_{OS}}{\partial \varepsilon}] = sign[(\sqrt{\beta} - \sqrt{(1-\beta)}) + \varepsilon(\sqrt{\beta} + \sqrt{(1-\beta)})] \qquad (10)$$

Thus, if there are more high ability programmers than low (i.e., $\beta \geq ½$), the quality improves with higher diversity among programmers (i.e., higher ε). On the other hand if the number of low ability programmers is very large compared to that of high quality, higher diversity reduces the quality. In the latter case, it is better if the programmer abilities are close to the average ability within this group.

5 CLOSED SOURCE QUALITY AND COMPARISON WITH OPEN SOURCE

In closed source software development, a dedicated team of programmers is maintained to design and improve proprietary software code. However, individual programmer effort and contribution quality are expensive to monitor, typical of principal-agent and team compensation models. As a result, compensation cannot be tied to individual performance metrics. Neither can a fixed salary be given since it would encourage employees to shirk. Therefore, programmer compensation is generally based on the total quality outcome. To motivate programmers to work harder, firms typically offer shares in the company's profits to employees (Hermalin 1998). This scheme is taken as the basis for the closed source firm. The following sequence of actions is considered:

Stage 1: The firm announces the compensation plan and hires M_{cs} programmers.

Stage 2: Each programmer chooses a quality output q_i. The combination of these individual outputs determines total software quality.

Stage 3: The firm sells the software at price P and realizes a profit.

This description of a closed source firm follows Hermalin closely, with some modifications. In Hermalin, a fixed number of risk neutral and identical team members shared the profit, while in the present case, a firm decides how many programmers to hire and how to share the profits with these programmers (i.e., all profits may not be shared). Further, employees are assumed to be risk averse. The game is solved by backward induction, solving the last stage first.

Stage 3 Analysis:

The demand for closed source software Δ_{CS} (Q_{CS}, P) is given by equation (2). The profit of the closed source firm is:

$$\Pi_{CS}(Q_{CS}) = \underset{P}{\text{Max}} \quad P\Delta_{CS}(Q_{CS}, P) \tag{11}$$

$$\Rightarrow P = \frac{-\Delta_{CS}(Q_{CS}.P)}{\partial\Delta(Q_{CS}, P)/\partial P}$$

The marginal cost of reproducing software is usually negligibly small and, therefore, is not included in (11).

Stage 2 Analysis:

The firm shares a proportion $(1 - \rho)$ of the total profit with employees and retains the rest. Let $C_{CS}(q_i)$ denote the cost of the programmer's effort in this closed source environment. $C_{CS}(q_i)$ is given by equation (4). Then the utility function is:

$$U_i = [u_i(1-\rho)\Pi_{CS}(Q_{CS})]^{\upsilon} - C_{CS}(q_i) \tag{12}$$

where programmer i receives a fraction u_i of the profit available to all programmers.

The employee's optimal quality choice is obtained by solving the first order condition.

Stage 1 Analysis:

In the first stage, the firm maximizes its share of the profit less variable expenses such as recruitment, coordination, overhead, and equipment costs, which is generally a convex function of number of employees.

$$\underset{\rho, M_{CS}}{\text{Max}} \quad [\rho\Pi_{CS}(Q_{CS})]^{\upsilon} - C(M_{CS}) \tag{13}$$

We assume that the firm's cost of overhead is given by $C(M_{CS}) = kM_{CS}^{\ 2}$.

With a heterogeneous pool of employees, it may be possible that the different employees are compensated differently using a mechanism design. This outcome is called a separating equilibrium, i.e., an equilibrium where the firm offers different amounts to the different programmer types. The following proposition shows this is not feasible.

Proposition 2: *Under the assumptions that individual programmer qualities are unobservable and the compensation is based on profit sharing, a separating equilibrium does not exist.*

Therefore, we examine a pooling equilibrium where all programmers obtain an equal share. The expected quality Q_{CS} in the closed-source model can now be obtained:

Proposition 3: *The ratio of programmers' quality levels* $\frac{q_H}{q_L} = \frac{\lambda_L}{\lambda_H}$. *Software quality obtained is*

$$Q_{CS} = \left(\frac{D^{(1+\upsilon)} \alpha^4 \theta^{(3+\upsilon)} \upsilon^{2(\upsilon+1)} (1-\upsilon)^{1-\upsilon} \left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L} \right]^2}{4^{(1+2\upsilon)} k^{1-\upsilon}} \right)^{\frac{1}{4-\theta\upsilon-\theta}}.$$

The number of programmers is endogenously determined and does not enter the quality expression. However, the quality of the closed source software improves with higher demand (i.e., higher D), better coordination (i.e., higher α), more high quality programmers (i.e., higher β), higher consumer valuation (i.e., higher θ), and lower costs (i.e., lower λ_H, λ_L, or k). The effect of the risk aversion is difficult to derive analytically.

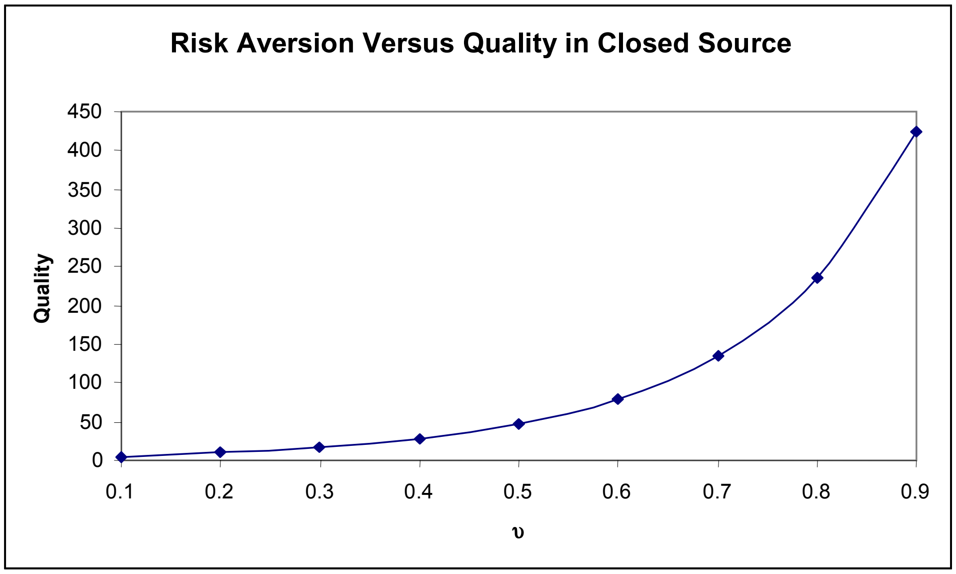

Figure 1. Effect of Programmer Risk Aversion on Quality

However, numerical simulations (see Figure 1 for the following parameters: D = 1000000, $\alpha = 1$, $\theta = 0.5$, $\beta = 0.5$, $\lambda_H = 1$, $\lambda_L = 2$, $k = 1$) show that higher risk aversion (i.e., lower ν) reduces quality. It also follows from Proposition 3 that if the coordination parameter a is zero or the risk aversion is very high, closed source quality is negligible.

Comparing overall quality under open and closed source environments, we get

Proposition 4: *The quality of open source software, Q_{OS},, is lower, equal to, or higher than the quality under closed source, Q_{CS}, if*

$$N_{OS}R(D)^{\upsilon}\begin{Bmatrix}<\\=\\>\end{Bmatrix}c\left(\frac{D^{(1+\upsilon)}\alpha^{4}\theta^{(3+\upsilon)}\upsilon^{2(\upsilon+1)}(1-\upsilon)^{1-\upsilon}\left[\frac{\beta}{\lambda_H}+\frac{1-\beta}{\lambda_L}\right]^{-\frac{\theta(1+\upsilon)}{2}}}{4^{(1+2\upsilon)}k^{1-\upsilon}}\right)^{\frac{2}{4-\theta\upsilon-\theta}}$$

Propositions 3 and 4 provide several qualitative insights that are likely to hold for other functional specifications:

First, there is no universal quality dominance of open source over closed source or vice versa under all circumstances. Second, the open source model becomes more attractive in an environment where a large pool of programmers is willing to work on open source software. This provides a rationale as to why the Internet revolution has been a major impetus for open source by allowing programmers worldwide to participate. Quality can be further enhanced if more programmers can be urged to participate, if programmers' efforts are publicly recognized, and if programmers perceive code contribution to be a fun task or a hobby.

Third, it is difficult to compare programmer ability in open source and closed source environments. We have taken β to be the same under both circumstances. It is possible that a closed source firm has a potential advantage in that a selection process may help identify and eliminate low ability programmers (i.e., only high ability programmers are employed despite a heterogeneous labor supply). On the other hand, it can be argued that open source programmers may be more skillful because they are self-selected. Also, since the source code is subject to extensive peer review, a level of personal accountability exists in open source development that does not exist in the closed source model (http://www.linuxworld.com/linuxworld/lw-1998-11/lw-11-ramparts.html). If the open source programmers are of higher quality compared to closed source, the results of our model will tilt in favor of open source quality.

6 CONCLUSIONS

In this paper, we examined software quality improvement under two possible models of software development: The traditional closed source model (e.g., Microsoft's Windows NT) and the emerging open source model (e.g., Linux).

We found that open source software quality is not necessarily lower than closed source quality despite the absence of a proprietary interest, and neither paradigm dominates under all circumstances. The quality of software developed by the open source tournament may sometimes but not universally dominate over the quality of closed source software depending on a combination of factors such as compensation, software demand, number of programmers, reward systems, programmer's cost of effort, and coordination of programmers.

However, it is still possible to compare marginal improvements in quality between open and closed source development due to each of the above factors, when all else is equal. First, quality is likely to improve in both open source and closed source with increasing market demand for the software. Both closed source firms and open source communities will have stronger incentives to improve software that are desired more by users, but for different reasons. Closed source firms see increased market demand as an opportunity for generating more revenues through sales and licenses, while open source programmers have a higher opportunity to gain recognition among their community or signal their talent to potential employers (open source or otherwise).

Since many open source programmers come from a hacker culture, they perceive identifying bugs and fixes as a fun or enjoyable activity, and hence increase quality by reducing programmers' cost of effort. The enjoyment associated with open source code improvement acts as an intrinsic motivator, enabling programmers to create and share improvements, even in the absence of any extrinsic rewards such as pay.

As a future extension we are currently examining the effect of the nature of the market (monopolistic versus competitive) on the quality of open and closed source software. Preliminary findings are that quality of both open source and closed source software decreases in competitive markets. This result suggests that competition from the open source movement may hamper the innovation in the closed source software industry and vice versa.

7 REFERENCES

Bloor Research. "Linux Versus NT: The Verdict," Press Release, October 22, 1999, Press Release (available online at http://www.bloor-research.com/product/245/linux_v_windows_nt_the_verdict.html).

Green, J. R., and Stokey, N. L. "A Comparison of Tournaments and Contracts," *Journal of Political Economy* (81), June 1983, pp. 349-365.

Hermalin, B. E. "Towards an Economic Theory of Leadership: Leading by Example," *American Economic Review*, 1998, pp. 1188-1205.

Holmstrom, B. "Moral Hazard in Teams," *The Bell Journal of Economics* (13), 1982, pp. 324-340.

Lazear, E. P., and Rosen, S. "Rank Order Tournaments as Optimum Labor Contracts," *Journal of Political Economy* (89), October 1981, pp. 841-864.

Lerner, J., and Tirole, J. "The Simple Economics of Open Source," Working Paper, Harvard Business School, Cambridge, MA, 2000.

Mood, A., Graybill, F. A., and Boes, D. C. *Introduction to the Theory of Statistics* (3rd ed.). Singapore: McGraw-Hill, 1974.

O'Reilly, T. "Lessons from Open-Source Software Development," *Communications of the ACM* (42:4), April 1999, pp. 32-37.

Prahalad, C. K., and Krishnan, M. S. "The New Meaning of Quality in the Information Age," *Harvard Business Review* (77:5), September/October 1999, pp. 109-118.

Raymond, E. S. "The Cathedral and the Bazaar," Working Paper, Thyrsus Enterprises, 1999 (available online at http://www.tuxedo.org/~esr/writings/cathedral-bazaar/).

Rosen, S. "Prizes and Incentives in Elimination Tournaments," *The American Economic Review* (76:4), September 1986, pp. 701-715.

Technical Appendix

Proof of Proposition 1

We relabel programmers so that what was earlier referred to as the i^{th} programmer is now the N^{th} programmer, and apply a statistical result (Mood et al. 1974, p.183) as an intermediate step:

Proposition 1*:* If $\{Q_1, Q_2, \ldots, Q_{N-1}\}$ are independent random variables with c.d.f.s $\{F_{Q1}, F_{Q2}, \ldots, F_{Q,N-1}\}$ respectively, and $Y_{N-1} = \max\{Q_1, Q_2, \ldots, Q_{N-1}\}$ then the c.d.f. of Y_{N-1} is $F_{Y_{N-1}} = \prod_{j=1}^{N-1} F_{Qj}$.

Proof of Proposition 1: $\text{Prob}[Y_{N-1} < Q] = \text{Prob}[Q_1 < Q] \cap \text{Prob}[Q_2 < Q] \cap \ldots \cap \text{Prob}[Q_{N-1} < Q] = \prod_{j=1}^{N-1} F_{Qj}$.

We observe that $p(q_i, q_{-i}) = \text{prob}[Q_i - Y_{N-1} \ ^3\ 0]$. In other words $p(.)$ is the c.d.f. of the random variable $(Y_{N-1} - Q_i)$ and can be determined using the convolution theorem, $p(Y_{N-1} - Q_i \leq 0) = \int_{-\infty}^{\infty} f_{Q_i}(x) \prod_{j=1}^{N-1} F_{Q_j}(x)dx$.

If $Y_N = \prod_{j=1}^{N} F_{Qj}$ is the distribution of the maximum quality, its expectation is obtained by the formula $E(Y_N) = \int_0^{\infty} [1 - F_{Y_N}] - \int_{-\infty}^{0} F_{Y_N}$ (Mood et. al 1974, p. 65). Q.E.D.

Proof of Corollary 1.1

Starting with equation (1) and (Tournament Winner Assumption)

$$U_i = \frac{q_i}{\sum_{j=1}^{N_{OS}} q_j} R(D_{OS})^{\upsilon} - C_{OS}(q_i) \tag{i}$$

Solving for quality chosen, the first order condition is:

$$\frac{R(D_{OS})^{\upsilon}[\sum_{j=1}^{N_{OS}} q_j - q_i]}{[\sum_{j=1}^{N} q_j]^2} - c\lambda_i q_i = 0$$

$$\Rightarrow \frac{R(D_{OS})^{\upsilon}[(N_H - 1)q_H + N_L q_L]}{[N_H q_H + N_L q_L]^2} - c\lambda_H q_H = 0 \quad \text{and} \quad \frac{R(D_{OS})^{\upsilon}[N_H q_H + (N_L - 1)q_L]}{[N_H q_H + N_L q_L]^2} - c\lambda_L q_L = 0$$

$$\Rightarrow \frac{R(D_{OS})^{\upsilon}}{(N_H q_H + N_L q_L)c\lambda_H} q_H \quad \text{and} \quad \frac{R(D_{OS})^{\upsilon}}{(N_H q_H + N_L q_L)c\lambda_L} q_L \tag{ii}$$

The approximation is reasonable if the input of a single programmer is small compared to the total quality. The second order condition is always met. Dividing the two equations we get $\lambda_L/\lambda_H = q_H/q_L$. Multiplying the first equation by N_H and the second by N_L and adding we get the expected quality of the software:

$$Q_{OS} = \sqrt{R(D_{OS})^{\upsilon}(\frac{N_H}{c\lambda_H} + \frac{N_L}{c\lambda_L})} = \sqrt{\frac{N_{OS}R(D)^{\upsilon}}{c}(\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L})} \text{ QED.} \tag{iii}$$

Proof of Proposition 2

Proof by contradiction. Suppose a separating contract (x_1, Q_{CS}) and (x_2 ,Q_{CS}) is announced where $x_1 > x_2$. These would satisfy the following incentive compatibility constraints:

IC$_1$: $\Pi_1(x_1, Q_{CS} \mid x_1) > \Pi_1(x_2, Q_{CS} \mid x_1)$

IC$_2$: $\Pi_2(x_2, Q_{CS} \mid x_2) > \Pi_2(x_1, Q_{CS} \mid x_2)$

However, IC$_2$ is violated since each employee prefers the higher share. Intuitively what this means is that since the firm can only observe the combined output of the employees, it cannot discriminate between the employees since each one would pretend to be of the high type without any cost incurred for lying. Q.E.D.

Proof of Proposition 3

We solve the game by backward induction, solving the last stage first. Stage 3 analysis gives:

$$\Pi_{CS}(Q_{CS}) = \underset{P}{Max}\, D(1-\frac{P}{Q^{\theta}_{CS}})P$$

$$\Rightarrow \Pi_{CS}(Q_{CS}) = \frac{DQ^{\theta}_{CS}}{4}$$

Substituting this into the Stage 2 analysis, we get the utility of the programmer.

$$\left[\frac{(1-\rho)DQ^{\theta}_{CS}}{4M_{CS}}\right]^{\upsilon} - \lambda_i \frac{q_i^2}{2}$$

We write the following two conditions for the two types of employees:

$$\frac{\alpha\theta\upsilon}{\lambda_H}\left[\frac{(1-\rho)D}{4M_{CS}}\right]^{\upsilon}(Q_{CS})^{\theta\upsilon-1} = q_L$$

$$\frac{\alpha\theta\upsilon}{\lambda_H}\left[\frac{(1-\rho)D}{4M_{CS}}\right]^{\upsilon}(Q_{CS})^{\theta\upsilon-1} = q_H$$

An immediate conclusion from these equations is that the efforts of the two types are inversely proportional to their cost parameter.

$$\frac{q_H}{q_L} = \frac{\lambda_L}{\lambda_H}$$

This implies that the low cost programmers choose a higher quality level than the high cost programmers. We now solve for software quality. Multiplying the first equation by αM_{H} and the second by αM_{L} and adding we solve as follows.

$$M\theta\upsilon\alpha^2\left[\frac{(1-\rho)D}{4M_{CS}}\right]^{\upsilon}\left[\frac{\beta}{\lambda_H}+\frac{1-\beta}{\lambda_L}\right] = (Q_{CS})^{2-\theta\upsilon}$$

$$\Rightarrow Q_{CS} = \left(M_{CS}{}^{1-\upsilon}\theta\upsilon\alpha^2\left[\frac{(1-\rho)D}{4}\right]^{\upsilon}\left[\frac{\beta}{\lambda_H}+\frac{1-\beta}{\lambda_L}\right]\right)^{\frac{1}{2-\theta\upsilon}}$$

Now, in Stage 1, the firm maximizes its share of the profit.

$$\underset{\rho, M_{CS}}{Max}\frac{\rho D\left(M_{CS}{}^{1-\upsilon}\theta\upsilon\alpha^2\left[\frac{(1-\rho)D}{4}\right]^{\upsilon}\left[\frac{\beta}{\lambda_H}+\frac{1-\beta}{\lambda_L}\right]\right)^{\frac{\theta}{2-\theta\upsilon}}}{4} - kM^2_{CS}$$

$$\Rightarrow \rho = 1-\frac{\theta\upsilon}{2}$$

$$\Rightarrow M = \left(\frac{D^2 \alpha^{2\theta} \theta^{2+\theta} \upsilon^{\theta(\upsilon+1)} (1-\upsilon)^{2-\theta\upsilon} \left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L} \right]^{\theta}}{2^{4+\theta\upsilon} k^{2-\theta\upsilon}} \right)^{\frac{1}{4-\theta\upsilon-\theta}}$$

Substituting these back into the quality expression we get:

$$Q_{CS} = \left(\frac{D^{(1+\upsilon)} \alpha^4 \theta^{(3+\upsilon)} \upsilon^{2(\upsilon+1)} (1-\upsilon)^{1-\upsilon} \left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L} \right]^{2}}{4^{(1+2\upsilon)} k^{1-\upsilon}} \right)^{\frac{1}{4-\theta\upsilon-\theta}} \qquad \text{Q.E.D.}$$

Proof of Proposition 4

The proof follows from the expressions of Q_{os} and Q_{CS}.