Open Source Versus Closed Source: Software Quality in Monopoly and Competitive Markets

Srinivasan Raghunathan, Ashutosh Prasad, Birendra K. Mishra, and Hsihui Chang

***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 softwares are inferior in quality compared to closed source softwares because of lack of incentives and project management, while proponents argue the opposite. This paper examines this quality debate by modeling and analyzing software quality, demand, profitability, and welfare under open and closed source environments in monopoly and competitive markets. The results show no dominant quality advantage of one method over another under all circumstances. Both open source and closed source qualities decrease in a competitive market. Conditions under which each method can generate higher quality software are examined.**

***Index Terms*—Competition, open source, software quality.**

I. 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 [1]. 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 [2]. Enhancements submitted by individual users are fed back to the original source code for public use. This approach has led to the development of the Linux operating system, Apache web server, Perl programming language, Sendmail electronic mail transfer agent, domain name system Berkeley internet name domain (DNS BIND) system, and other Internet infrastructure software that power some of today's most powerful electronic commerce websites including Yahoo, Cisco Systems, and C-Net. Table I provides short descriptions of some commonly used open source software.

Manuscript received November 5, 2003; revised May 28, 2004 and September 20, 2004. This paper was recommended by Associate Editor H. R. Rao.

S. Raghunathan and A. Prasad are with the School of Management, University of Texas at Dallas, Richardson, TX 75083 USA (e-mail: sraghu@utdallas.edu; aprasad@utdallas.edu).

B. K. Mishra is with the Anderson Graduate School of Management, University of California at Riverside, Riverside, CA 92521 USA, on leave from the School of Management, University of Texas at Dallas, Richardson, TX 75083 USA (e-mail: barry.mishra@ucr.edu).

H. Chang is with the Anderson Graduate School of Management, University of California at Riverside, Riverside, CA 92521 USA (e-mail: hsihui.chang@ucr.edu).

Digital Object Identifier 10.1109/TSMCA.2005.853493

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") is based on the assumption that software development is 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 [3], open source represents a "bazaar style" of software development, in contrast to the "cathedral style" emphasized by closed source software development.

One of the frequent criticisms of open source softwares is that they are of lower quality compared to their closed source counterparts. First, critics contend that the quality of open source software suffers from the "free rider" effect often associated with public goods, where each user prefers others to spend effort in improving the public good and then share in the benefits rather than investing in improvements themselves. Thus, improvement depends on the altruism of a few. Furthermore, attempts at commercial exploitation of freely donated software can result in an amelioration of altruistic tendencies [4]. However, proponents claim that software contributors are not necessarily altruistic; programmers contribute to the source code for social recognition and prestige in the open source community and to signal their talent to prospective employers, which may subsequently translate into job, consulting, or other career opportunities [5]–[7]. A discussion of possible alternative motivations of open source programmers is provided by [8]. The focus here will be on pecuniary, or economic, motives as opposed to altruistic motives [7].

Second, 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, and is hence more reliable. Limited coordination in an open source environment may lead to some duplication of effort between programmers trying to

TABLE I
EXAMPLES OF OPEN SOURCE SOFTWARE

	Description	Closed Source Competitors
Linux	Based on a kernel designed by Linus Torvalds, 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, OpenBSD, and other Berkeley UNIX derivatives boast an estimated 1 million users.	Solaris (Sun), HP-UX (HP)
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)
Tcl	John Ousterhout's tcl language is used by over 300 000 users.	
Python	Guido van Rossum's Python language is used by over 325 000 users.	
GNU project	A high-quality set of programming tools, including the gcc C compiler, g++ C++ compiler, emacs editor, and gdb debugger from the Free Software Foundation's GNU project.	Visual C++ (Microsoft), Cold Fusion (Allaire)
Patch	Larry Wall's patch program allows users to exchange small fixes to programs in isolation rather than exchanging the entire software.	
CVS	Concurrent versioning system (CVS) allows users to maintain multiple versions of the same software.	
Apache	Originally created by Rob McCool in 1995, Apache is currently deployed in over 53% (1 million) of today's web servers, well ahead of second-placed Microsoft's IIS at 23%.	Internet Information Server (Microsoft)
Mozilla	Netscape's next-generation web browser Mozilla is the open source version of its popular browser product Communicator.	Internet Explorer (Microsoft)
BIND	Berkeley Internet name daemon servers are the core of the Internet's domain name service (DNS) system.	
Sendmail	Developed by Eric Allman as part of Berkeley UNIX, sendmail is the primary e-mail transport/forwarding agent on the Internet, routing e-mails in intermediate nodes between sending and receiving nodes. Used by 80% of all Internet sites.	
Samba	Designed by Andrew Tridgell, Samba allows UNIX and Linux systems to act as file and print servers on NT and Windows 95/98 networks.	
Ghost-script	Postscript editor and printer	Adobe Acrobat
Open-Office. Org	Sun Microsystems' open source word processor, spreadsheet and other general-purpose office application software.	Microsoft Office

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. While process maturity concepts such as the capability maturity model (CMM) are not directly applicable to open source projects, elements of these approaches such as processes for code submission, peer review, and user involvement are strictly followed, particularly in larger projects, which also maintain good documentation.

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 [9]. A related issue of interest is whether the society as a whole is better off under the closed source or the open source paradigm. The free nature of open source software may benefit consumers, but if the quality is low, then the value consumers derive from the software will also be low. From a society perspective, the profit realized by the closed source software developed is also important.

The objectives of this research are threefold: 1) to analyze the impact of inherent incentive structures of the closed and open source approaches on software quality; 2) to determine whether the open source or closed source approach always results in a superior software quality and, if not, to identify the conditions under which each approach is better; and 3) to compare the societal welfare, defined as the sum of developer profit and consumer surplus, in the open source and closed source paradigms. The game theory was used to perform the analysis. Software quality is analyzed in noncompetitive open source and closed source markets, and then in a competitive market where both open source and closed source softwares compete for market share.

The 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 the closed source quality. The open source model becomes attractive when 1) a large pool of programmers is willing to work on the open source software; 2) the programmers' efforts are publicly recognized; and 3) open source programmers perceive code contribution to be a fun task or a hobby. If the open source and closed source softwares have the same quality, then the social welfare is higher under the open source than the closed source approach. If a closed source software and an open source software compete in the same market, then the open source quality is independent of closed source software quality but is lower than the quality when there is no competition. However, under competition, the closed source quality decreases as the open source quality increases. The social welfare is higher under competition than in a monopoly.

For purposes of model tractability, several factors governing the closed source and open source approaches are ignored in this paper. Demand for software may depend on factors such as functionality, standards, support, and legal issues. In the proposed model, demand is a function of only quality and price. While factors such as standards and licensing arrangements are also important considerations in a firm's decision to adopt the software, these factors are covered to the extent they impact utility for the software, in which case they fall under the rubric of "quality."

The remainder of the paper proceeds as follows. Section II reviews quality debate surrounding software development, specifically focusing on open source software. Section III discusses the modeling framework and assumptions. Section IV examines how quality improvement comes about in a monopoly market with only one open source software. Section V

investigates quality improvement when closed source software is in monopoly and compares the result to those in the preceding section. Section VI models market competition between open source and closed source softwares. Section VII concludes with discussion and directions for future research.

II. The Quality Debate in Open Source Software

Software quality has been extensively studied in software engineering. In this literature, the quality of a software is measured along dimensions such as correctness, understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, structuredness, and efficiency [10], [11]. Despite advances in software process innovations such as object-oriented analysis and design, function point analysis, and CMMs, the software industry has continually been plagued by an inability to generate quality software products [12], [13]. It is noted in [13] that, "In the last 15 years alone, software defects have wrecked a satellite launch, delayed an airport opening for a year, destroyed a Mars mission, killed four Marines in a helicopter crash, induced a U.S. Navy ship to destroy a civilian airliner, and shut down ambulance systems in London leading to as many as 30 deaths."

The complex software systems that have become the central nervous system of firms make the high quality of a software vital. They not only control a firm's everyday operations but also enable or constrain the ability to make and implement strategic business decisions [14]. Though software quality investments can reduce overall software life cycle costs by minimizing rework later on, many software manufacturers sacrifice quality in favor of other objectives such as shorter development cycles and meeting time constraints [15]. As one software manager said, "I'd rather have it wrong than have it late. We can always fix it later" [15]. The traditional conception of software quality is centered on a product-centric conformance view of quality [14]. The focus of this perspective is to reduce variability (by minimizing errors in design and coding) and to conform to predefined specifications (e.g., in terms of error rates, speed, transaction load, etc.). 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 key features of a software while others may consider ease of installation and maintenance as key features of the same software. The authors will assume that howsoever quality is described, it may be measured from the customers' viewpoint as a utility providing attribute such that a higher quality product is preferred by all users.

The authors can make a qualitative comparison of how quality might vary between open source and closed source softwares. Open access to the source code allows greater opportunities for customizing an open source software to the specific needs of individual users than a closed source software. In fact, such customization is encouraged and widely practiced by the open source community, while closed source softwares that provide clients with access to the source code (e.g., SAP R/3) proclaim that any modification to the source code will result in loss of customer support, thus limiting the product's customizability to a predefined set of options. Furthermore, 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. In practice, upgrades to open source software are released much more frequently than that of closed source software, and consequently, open source software such as Linux seems to have developed faster than comparable closed source software such as Windows NT. As an example, Linux 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 softwares 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 [16] 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 less hardware configuration requirement, lightweight nature, better multiprocessing capabilities, and networking support [17]; however, others prefer Windows NT because of better security, ease of management, and superiority in Java performance.

The issue of cost comparison between an open source and a closed source has also been of interest to organizations, but there are few academic studies available on precise and objective comparisons. It is still a new issue for academics. Among open source softwares, Linux is the one that has received the most attention. A few reports exist comparing it to Windows, but the results are inconclusive. For example, following is the abstract of an IDC report [18] sponsored by Microsoft on the total cost of ownership (TCO):

> Linux is widely regarded as "free" because there is no or little cost associated with software acquisition. However, after taking into account all costs, notably IT staffing, does Linux truly come at a lower cost than competing platforms, such as Windows? IDC has completed a study of five common workloads in enterprise computing that challenges the common industry perception that Linux is "free." Our in-depth study suggests that Microsoft Windows 2000 offers lower total cost than a Linux solution

TABLE II
COST OF OWNERSHIP DATA (IDC 2002)

	Networking		File		Print		Security		Web	
	Microsoft	Linux	Microsoft	Linux	Microsoft	Linux	Microsoft	Linux	Microsoft	Linux
Hardware	1211	1004	5703	3139	1173	2172	1653	2041	7087	3006
Software	211	940	3988	1009	1665	340	5829	6609	7107	1390
Staffing	8392	8201	54 030	81 204	40 247	59 080	50 609	71 056	15 102	23 015
Downtime	1412	1494	30 133	20 788	38 857	39 746	10 335	4385	1646	1541
IT Staff Training	534	677	5191	7670	4787	5282	2000	6445	1304	1584
Outsourced	26	946	3	570	121	369	49	440	59	64
Total	11 787	13 263	99 048	114 381	86 849	106 989	70 495	90 975	32 305	30 600

Windows 2000 and Linux Server Environment Five year total cost of ownership summary view for 100 supported users by workload and cost category ($) as reported by IDC 2002

in four of the five workloads common to most corporate IT environments. In these four workloads (network infrastructure, print serving, file serving, and security applications), the cost advantages of Windows are significant: 11–22% less over a 5-year period. The cost advantages are driven primarily by Windows' significantly lower costs for IT staffing, generally the largest single component of IT costs. For the fifth workload, Web serving, Linux had a cost advantage of 6% compared with Windows 2000 over the 5-year period. IDC's study confirms that low initial software acquisition costs are only one factor, not the deciding one, in determining the 5-year total cost of ownership (TCO) for the two operating environments.

The data reported by IDC are given in Table II.

Contrary conclusions are reported by [19], as shown in Table III. This also takes into account software, hardware, support, and maintenance costs, and found that Linux was the least expensive platform to deploy and operate. Although some initial costs were higher at points, the ability to massively scale the product horizontally without paying additional licensing fees can yield significant cost savings over the long term.

The lack of jury on the open source versus closed source debate is reflected in the fact that closed source firms such as IBM and Sun 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, the enhanced version is distributed 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.

TABLE III
COST OF OWNERSHIP DATA (ROBERT FRANCES GROUP)

	Year 1	Year 2	Year 3
Linux	$49 931	$62 203	$74 475
Solaris	$421 718	$491 619	$561 520
Windows	$91 724	$141 193	$190 662

Finally, as suggested by the open source examples listed in Table I, 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. The 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. Further, open source programmers are less likely to contribute to highly specialized applications because the small user base for such software limits the amount of rewards programmers can derive by contributing code.

The authors summarize this section with Table IV listing the key features and relevant differences between open source and closed source softwares. These features provide the conceptual underpinnings for modeling open source and closed source software development within an economic framework. Also note that open source software development resembles a tournament [20], [21] whereas closed source development resembles the standard production model of the firm. Using this intuition, the authors proceed to modeling software quality developed under the two methods.

III. MODELING FRAMEWORK AND ASSUMPTIONS

The modeling framework is based on models of quality and price-based competition in a vertically differentiated market.

TABLE IV
CONCEPTUAL UNDERPINNINGS OF THE MODELING FRAMEWORK

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

TABLE V
LIST OF MAIN NOTATION

Symbol	Quantity
$V = tQ^{\theta}$	Value to consumers
$Q \quad (Q_{\text{os}}, Q_{\text{cs}})$	Quality of the software (open source, closed source)
$Q_i = q_i + \varepsilon_i$	Individual developers quality
F_{Q_i}	Distribution of Q_i
α	Coordination parameter
$C(q_i) = c\lambda_i q_i^2/2$	Cost of quality
$\beta \in [0,1]$	Proportion of high cost programmers
$w(y) = y^{\nu}$	Utility of a programmer
$\nu \in (0,1)$	Risk aversion parameter
$N_{\text{os}}, M_{\text{cs}}$	Number programmers (open source, closed source)
D	Market size
$S(Q,P)$	Demand for software of quality Q at price P
$R(D)$	Reward for the winning open source programmer
q_i	Planned quality choice by the individual developer
P	Price of the closed source software
ρ	Fraction of total profit retained by the firm

Quality and price-based competition have been analyzed in a variety of settings such as service delivery [22]–[24], marketing and supply chain channels [25], and economics. The primary difference between this paper and prior work in this area is that the authors model the impact of incentive structure on software quality whereas prior research assumes that firms can set any quality level they desire. This section presents a software development model from both consumer and developer perspectives. Table V provides a summary of the notation used in this paper.

On the consumer side, the authors assume that the consumer valuation for the software at quality Q takes the form

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

The parameter $\theta \in (0,1)$ captures the declining marginal value of quality. That is, a consumer's value for a software is a concave function of its quality. The model assumes a scalar measure of quality, which could be any one or an aggregate of some or all of the software quality measures discussed in Section II. If the heterogeneity parameter t is uniformly distributed between 0 and 1, the demand function is obtained for the software at quality Q and price P, given a total of D consumers

$$S(Q,P) = D\left(1 - \frac{P}{Q^{\theta}}\right). \tag{2}$$

On the developer side, the software is developed by a set of programmers. Programmer i chooses a planned quality output q_i by applying the requisite effort. However, software improvements need not follow deterministically from the amount of effort applied. Improvements may come randomly, in a serendipitous manner, by "stroke of luck." Additionally, some randomness is inherent in the subjective nature of the evaluation process. Therefore, the evaluated quality Q_i is modeled as a random variable with a deterministic component (function of programmer effort) and a random term. Thus

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

The cumulative distribution function (cdf) of Q_i is denoted F_{Q_i}. Programmers' costs $C(q_i)$ are assumed to be increasing and convex, i.e., $C'(q_i) > 0$ and $C''(q_i) > 0$. The cost of effort is typically considered to be convex by economists. Compared to spending the first hour on a project, spending an additional hour after having spent say 8 h is more difficult due to physical and mental fatigue and the press of other duties. Also, initial improvements are easier to make because they are more obvious—like low-hanging fruit—compared to finding later improvements. Among convex functions, the quadratic form is selected for its tractability.

Programmers differ in their skill and experience. Presumably, all programmers can ultimately get a job done, but the more skilled and experienced ones will find it less costly in terms of time and effort to do so. One can thus differentiate programmer types by their cost of programming parameter. For simplicity, the authors assume two types, high cost (H) or low cost (L); however, this simplification is not critical to the results here. The authors denote the number of programmers in high- and low-cost 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 and $\beta \in [0,1]$ is the proportion of high-cost programmers.

For the same planned output quality, a low-type programmer incurs a lower cost compared to a high type. More specifically, the programmer's cost of quality is specified as

$$C(q_H) = \frac{c\lambda_H q_H^2}{2} \quad \text{or} \quad C(q_L) = \frac{c\lambda_L q_L^2}{2} \tag{4}$$

depending on the programmer's type. In this specification, c is a cost parameter normalized to 1 for the closed source model and $\lambda_H > \lambda_L > 0$ indicates the relative costs for the high- and low-cost programmers. The parameter c will likely take a value of less than 1 in the open source model. This is because open source programming is viewed by some programmers as a hobby or it may be done for altruistic motives [7].

The software's aggregate quality is proportional to the sum of the quality choices of the participating programmers. Participation can vary from a handful in small projects to hundreds of programmers such as in the case of the Apache web server [26]. Not all of them gain economical rewards. In a study of the Apache web server project, [6] concluded that higher status in a merit-based ranking of contributors is associated with significantly higher wages. Although, in the model here, only the top-ranking programmer obtains a reward, it can be modified so that there are rewards for several top-ranking contributors.

The effective overall quality is $\alpha \sum q_i$, i.e., the sum of individual qualities multiplied by a coordination parameter. The reason for summing qualities is that large programs are written in a modular manner. Modern programming languages such as C++ and Java are well suited for collaborative environments. The reason is that they use object-oriented programming so that the program is modular, and modules can be independently written, tested, and modified. In addition, there exist large libraries of programs that can be incorporated into the current project, greatly reducing programming time. Another feature of these languages is their error handling capabilities. On encountering an error, the entire program does not hang, but it gives the signal that an error has been detected so that it can be debugged. The user obtains utility from the parts of the program that work but is denied utility from features that are buggy. The authors assume that the net utility, which is a measure of the overall quality of the program, is the sum of these utilities, which is to say that the overall quality is the sum of the individual qualities.

The proportionality constant α accounts for the coordination between programmers. Thus, when lack of coordination leads to overlap of programming efforts, the overall quality is not improved to the same extent as when the effort is coordinated. Closed source software development encourages and facilitates close collaboration between programmers. The reason why open source projects might be less coordinated is that the contributing programmers are located in different locations and operate on a voluntary basis. They have different motivations and capabilities. As a result, the open source project generally works without explicit planning or schedules. However, coordination might not be too poor either. In particular, most Open Source projects have a team leader who coordinates the volunteers, assigns tasks and selects submissions for improvements, and invests a lot of time in management. Larger projects, such as Apache, have a formal core development team [26]. Furthermore, there is some formalization from the fact that many Open Source projects are listed on http://SourceForge.net or similar hosts, which provide tools such as concurrent versioning system (CVS), mailing lists, and discussion forums. These may be considered as *ad hoc* coordination mechanisms [27].

The authors normalize the value of α to 1 in the case of an open source software, while for a closed source software, α is likely greater than 1 because the closed source approach is designed to achieve a coordinated and synergistic outcome. However, from a modeling perspective, the authors allow the coordination cost parameter to take any value of fraction or multiple. This implies that Open Source coordination can be greater than or lower than in closed source.

The utility specification for programmers, for a reward amount y, is given by

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

where $\nu \in (0,1)$ is the risk aversion parameter. An individual or entity is defined to be risk neutral, risk averse, or risk seeking if the marginal utility from its payoff is a constant, decreasing, or increasing function, respectively [28]. The power function specification used here, and earlier for modeling the consumers' utility, is widely used in the economics and business literature for capturing risk aversion, e.g., [29].

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

Next, the authors compare the qualities of open and closed source softwares in monopoly and competitive markets.

IV. Quality Under Open Source

The open source model is examined first in a noncompetitive market. The discussion of the noncompetitive market is important. First, it provides a baseline for comparing competitive results. Second, software markets tend to exhibit preferences towards single dominant firms. Finally, in an early state of software development, the software is often sheltered or isolated from competition within a niche market of innovators and early adopters. The authors assume that the open source environment consists of a set of $N_{\mathrm{OS}} \geq 2$ programmers that submit improvements to the open source code. A coordination committee costlessly evaluates submissions 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 in a noncompetitive market and free, the entire market may be considered adopters of the software, so that D_{OS} is equal to the size of the market D.

The winning programmer obtains a reward valued at $R(D)$ with no reward for the rest of the field. In the event of a tie, the winner is randomly chosen. The 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 [7]. Lerner and Tirole [7] propose that open source programmers have two incentives to participate: the ego gratification incentive and the career concern incentive. The latter is a pure monetary concern and refers to future job offers, and shares in commercial open source companies. The ego gratification incentive stems from a desire for peer recognition. Both incentives increase with the number of users of the open source software; therefore, since reward is a function of the total number of users of the open source software, $R(D)$ can be interpreted as pertaining to both ego gratification and career advancement. The value of this reward increases with the size of the user base, i.e., $R'(D) > 0$.

Programmer i submits an entry with the expected payoff

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

where $p(\cdot)$ is the probability of the programmer winning the tournament and q_{-i} is a vector of the endogenously determined planned qualities of the remaining submissions. The probability of any programmer winning the tournament is the probability that their entry is evaluated better than the remaining $N_{\mathrm{OS}} - 1$ entries, i.e., for programmer i

$$p(q_i, q_{-i}) = \mathrm{prob}\left[q_i + \varepsilon_i \geq \max\{q_j + \varepsilon_j\}_{j \neq i}\right]. \tag{7}$$

To evaluate this probability, the authors need to specify the distribution of $\max\{q_j + \varepsilon_j\}_{j \neq i}$ (details given in the Appendix). Having done so, the authors can substitute (7) into the programmer's utility function and obtain the quality choice of the programmer and the expected software quality. Proposition 1 summarizes the results.

Proposition 1: The optimal planned quality for each programmer q_{OS} is the solution of the N_{OS} equations

$$\frac{\partial p(q_i, q_{-i})}{\partial q_i} R(D)^{\upsilon} - \frac{\partial C_{\mathrm{OS}}(q_i)}{\partial q_i} = 0, \quad i \in [1, N_{\mathrm{OS}}].$$

The expected quality of the winning software is $\int_0^{\infty}[1 - F_{Y_N}] - \int_{-\infty}^{0} F_{Y_N}$, where $F_{Y_N} \equiv \prod_{j=1}^{N_{\mathrm{OS}}} F_{Q_J}$ and F_{Qj} is the cdf of Q_j (proofs given in the 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, the authors make the following assumption.

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

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

This probability can be derived from Proposition 1 with a specification of the random variable based on the Gumbel distribution [30]. The above assumption has been used in the tournament literature [21], the literature on patent races [31], and the choice literature [30], [32]. Further, it is extensively used in competitive models as the multiplicative competitive interaction (MCI) model [33].

Although it is intuitive that high-cost programmers have a lower probability of winning, they do not voluntarily drop out. The first-order condition is

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

And note that if $q_i = 0$ in this expression, indicating the programmer dropped out, the left-hand side remains positive, indicating that the condition is not satisfied. Thus, the solution is bounded away from zero for all programmer types, implying participation by both high- and low-cost types. Equations for the high- and low-cost programmers can be written and solved to yield Corollary 1.1.

Corollary 1.1: From Proposition 1 and the tournament winner assumption, the ratio of high- and low-cost programmers' planned qualities is $q_H/q_L = \lambda_L/\lambda_H$. The quality of the software is $Q_{\mathrm{OS}} = \sqrt{(N_{\mathrm{OS}}R(D)^{\upsilon}/c)((\beta/\lambda_H) + (1-\beta/\lambda_L))}$. Social welfare is $DQ_{\mathrm{OS}}^{\theta}/2$ (proof given in the Appendix).

This result leads to the following interesting insights.

1) From Corollary 1.1, the quality improvement Q_{OS} is increasing in ν, N_{OS}, and D, and decreasing in β and c. Lower risk aversion leads to higher quality levels because the reward has greater expected utility for the programmers. A greater number of programmers and a larger fraction of highly productive programmers increase quality because of the higher cumulative quality. Improvements due to D follow from the logic that if reward from effort is increased, programmers are motivated to work harder.
2) 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, the model here suggests that such hobby considerations decrease the programmer's cost of effort, and thereby, increase the quality of open source software.
3) The role of N_{OS} in quality improvement can have two effects. Larger numbers of programmers decrease the likelihood of any individual obtaining the reward and, therefore, act as a disincentive to effort. At the same time, more programmers will provide a larger number of improvements that cumulatively lead to higher quality software. Since the quality expression is increasing in the number of programmers, the latter is the dominant effect.
4) 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 a

quality threshold Q^*, the expected reward is $p(Q_{\rm OS} \geq Q^*)p(q_i, q_{-i})R(D)^{\upsilon}$, not $p(q_i, q_{-i})R(D)^{\upsilon}$. Although the analytical solution is intractable, it can be seen that a quality threshold is detrimental to the 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, leading to a death spiral.

5) A question related to programmer heterogeneity is whether open source quality can be improved if there is a more heterogeneous population of programmers. By setting $\lambda_H = 1 + \varsigma$ and $\lambda_L = 1 - \varsigma$ and examining the sensitivity of quality to ς, it can be seen that

$$\operatorname{sign}\left[\frac{\partial Q_{\rm OS}}{\partial \varsigma}\right] = \operatorname{sign}\left[\left(\sqrt{\beta} - \sqrt{(1-\beta)}\right) - \varsigma\left(\sqrt{\beta} + \sqrt{(1-\beta)}\right)\right]. \quad (10)$$

Thus, if there are more high-cost than low-cost programmers (i.e., $\beta > 1/2$), the quality can improve with greater difference in programming costs between programmer types (i.e., higher ς). On the other hand, if the proportion of high-cost programmers is small, greater cost differences reduce the software quality. In the latter case, it is better if programmer abilities are close to the average ability within the group.

V. Closed Source Quality and Quality Comparisons

In closed source software development, a dedicated team of programmers is maintained to design and improve proprietary software code. However, the effort input by individual programmers is expensive to monitor, as discussed in the principal-agent literature in economics (e.g., [34]). Furthermore, it may be the case that only team outcomes are observable but not individual outcomes. It is well documented in the literature on team compensation that precisely knowing the effort expended by members of the team is difficult for an outside manager [35]. On the other hand, depending on inside members to regulate and report upon each other creates even more complicated incentive mechanisms. This lack of observability, for example, creates the well-known problem of free-riding, internal politics, trying to take credit for other's work, and the myriad of other problems that managers have to deal with. Second, within a closed source environment, programmers collaborate closely and the tasks cannot be made entirely independent. Itoh [36] has noted that a principal may want to reward an agent as a function of others' outputs, even if the outputs are independently distributed, when teamwork is desirable.

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 team's total quality outcome to motivate programmers to work harder. For example, sharing of profits through stock option plan and bonus is a common practice in many software development firms [37]. This scheme is taken as the basis for the closed source model. The authors made the assumption that programmers who develop the closed source software get a share of the profits because of two reasons. First, sharing of profits through stock option plan and bonus is a common practice in many software development firms. Second, this assumption enables to tie software quality to reward obtained by programmers.

The following sequence of actions is considered.

Stage 1) The firm announces its compensation plan and hires $M_{\rm CS}$ programmers.

Stage 2) Consistent with the open source model, each programmer chooses a planned quality q_i and the combination of individual quality outputs determines the total software quality.

Stage 3) The firm sells the software at a price P and realizes profits.

This description of the closed source firm follows [37] with the following generalizations. In [37], a fixed number of risk neutral and identical team members share the profit. In the present case, a firm decides how many programmers to hire and how much profit to share with risk averse and heterogeneous programmers.

With a heterogeneous pool of employees, it may be possible that the different employees are compensated differently using the mechanism design. This outcome is called a separating equilibrium, i.e., an equilibrium where the firm offers different amounts to different programmer types. The following proposition shows that 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 (proof given in the Appendix).

Therefore, the authors examine a pooling equilibrium where all programmers obtain an equal share. The game is solved by backward induction, solving the last stage first.

Stage 3 analysis: The demand for closed source software $S_{\rm CS}(Q_{\rm CS}, P)$ is given by (2). The profit of the closed source firm is

$$\Pi_{\rm CS}(Q_{\rm CS}) = \max_{\rm P} \; PS_{\rm CS}(Q_{\rm CS}, P)$$
$$\Rightarrow P = \frac{-S_{\rm CS}(Q_{\rm CS}, P)}{\frac{\partial S_{\rm CS}(Q_{\rm CS}, P)}{\partial P}} = \frac{Q^{\theta}_{\rm CS}}{2}. \quad (11)$$

The marginal cost of reproducing software is usually negligibly small and, therefore, excluded from (11). However, if there is a positive marginal cost of production, it can be scaled out without affecting the intuition.

Stage 2 analysis: The firm shares a proportion $(1 - \rho)$ of the profit with employees and retains the rest. Let $C_{\rm CS}(q_i)$ denote the cost of the programmer's effort in the closed source

environment. $C_{\mathrm{CS}}(q_i)$ is given by (4). The utility function for programmer i is

$$U_i = \left[\frac{(1-\rho)\Pi_{\mathrm{CS}}(Q_{\mathrm{CS}})}{M_{\mathrm{CS}}}\right]^{v} - C_{\mathrm{CS}}(q_i) \tag{12}$$

where programmer i receives a fraction $1/M_{\mathrm{CS}}$ of the profit available to all programmers. The employee's optimal quality choice is obtained by solving the first-order condition (steps given in the Appendix).

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

$$\underset{\rho, M_{\mathrm{CS}}}{\mathrm{Max}} \ [\rho\Pi_{\mathrm{CS}}(Q_{\mathrm{CS}})]^{v} - C(M_{\mathrm{CS}}). \tag{13}$$

The authors assume that $C(M_{\mathrm{CS}}) = kM_{\mathrm{CS}}$, where k is a constant representing the cost for a one person change in M_{CS}. The expected quality Q_{CS} in the closed source model can now be obtained.

Proposition 3: The ratio of the programmers' quality levels $q_H/q_L = \lambda_L/\lambda_H$. The price of the software is $Q_{\mathrm{CS}}^{\theta}/2$, the quality is

$$Q_{\mathrm{CS}} = \left(\frac{D^{\frac{2}{\theta}}\alpha^2\theta^{1+\frac{1}{\theta}}v^{v+1}(1-v)^{1-v}\left[\frac{\beta}{\lambda_H}+\frac{1-\beta}{\lambda_L}\right]}{8k^{1-v}}\right)^{\frac{1}{2-\theta}}$$

demand is $D_{\mathrm{CS}} = D/2$, and social welfare is given by $3DQ_{\mathrm{CS}}^{\theta}/8$ (proof given in the Appendix).

This result leads to the following interesting insights.

1) 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), and lower costs (i.e., lower λ_H, λ_L, or k).
2) The effect of the risk aversion is difficult to derive analytically. However, numerical simulations (see Table VI for the parameters: $D = 100$, $\alpha = 1$, $\theta = 0.5$, $\beta = 0.5$, $\lambda_H = 2$, $\lambda_L = 1$, $k = 1$) show that higher programmer risk aversion (i.e., lower ν) reduces quality. (The authors performed extensive numerical analysis for various parameter values. The results in all these simulations were qualitatively similar to that presented in Tables VI and VII.)
3) The effect of the consumer valuation parameter θ is also difficult to derive analytically. Numerical simulations show that a higher value of θ reduces the quality but increases firm's profit and welfare (see Table VII for the parameters: $D = 100$, $\alpha = 1$, $\nu = 0.9$, $\beta = 0.5$, $\lambda_H = 2$, $\lambda_L = 1$, $k = 1$).

Comparing the overall quality under open and closed source environments, the authors get Proposition 4.

TABLE VI
EFFECT OF ν ON QUALITY, PROFIT, AND WELFARE

ν	Quality	Profit	Welfare
0.10	1928.12	8.23	1646.64
0.20	2723.07	9.78	1956.87
0.30	3314.91	10.80	2159.07
0.40	3852.75	11.64	2327.64
0.50	4411.11	12.45	2490.61
0.60	5048.53	13.32	2664.49
0.70	5831.62	14.32	2863.69
0.80	6861.72	15.53	3106.33
0.90	8342.47	17.13	3425.14

TABLE VII
EFFECT OF θ ON QUALITY, PROFIT, AND WELFARE

θ	Quality	Profit	Welfare
0.10	4.2E14	6.89	1087.43
0.20	1.28E18	9.41	1567.75
0.30	624 628.86	11.64	2054.56
0.40	41 701.46	14.10	2642.97
0.50	8342.47	17.13	3425.14
0.60	2961.43	21.18	4538.71
0.70	1484.69	26.98	6225.54
0.80	938.61	35.82	8954.08
0.90	704.81	50.30	13 718.12

Proposition 4: In noncompetitive markets, the open source software quality is higher, equal to, or lower than the closed source software quality if

$$N_{\mathrm{OS}}R(D)^{v}\left\{\begin{matrix}>\\=\\<\end{matrix}\right\}$$
$$c\left(\frac{D^{\frac{2}{\theta}}\alpha^2\theta^{1+\frac{1}{\theta}}v^{v+1}(1-v)^{1-v}}{8k^{1-v}}\right)^{\frac{2}{2-\theta}}\left[\frac{\beta}{\lambda_H}+\frac{1-\beta}{\lambda_L}\right]^{\frac{\theta}{2-\theta}}.$$

Social welfare is higher, lower, or equal if

$$Q_{\mathrm{OS}}^{\theta}\left\{\begin{matrix}>\\=\\<\end{matrix}\right\}0.75Q_{\mathrm{CS}}^{\theta}$$

(proof given in the Appendix).

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

1) There is no universal quality dominance of open source over closed source and *vice versa* under all circumstances.
2) The open source model becomes more attractive in an environment where a large pool of programmers is willing to work on open source software. The Internet revolution has been a major impetus for open source by allowing programmers worldwide to participate. Quality can be further enhanced a) if more programmers can be urged to participate; b) if the programmers' efforts are publicly

recognized; and c) if programmers perceive code contribution to be a fun task or a hobby.

3) It is difficult to make prior assertions about programmer ability in open source versus closed source environments. Thus, the authors took β to be the same under both circumstances in the analysis. One could argue the possibility that a closed source firm has a potential advantage in that its hiring process may identify and exclude low-ability programmers so that only high-ability programmers are employed despite a heterogeneous labor pool. On the other hand, it can be argued that open source programmers are 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. If the open source programmers are of higher quality compared to closed source, the results of the model here will tilt in favor of the open source quality.
4) Social welfare calculations reveal that for the same quality of software, open source generates greater social welfare.

If an open source software provides greater quality than a closed source software, an interesting issue to consider is whether a closed source firm such as Microsoft can improve software quality by internally adopting an open source strategy of a tournament, where programmers work independently on the same problem and compete for a single reward. This means that the programmers do not divide their share of profit equally; rather, only one from the M programmers will get the entire profit shared by the firm. In Corollary 4.1, the authors provide a sufficient condition for when it is suboptimal for the firm to use a tournament approach if the benefit of programmer coordination is sufficiently high, i.e., α is sufficiently high.

Corollary 4.1: Let M^* be the optimal number of programmers hired by the closed source firm under the tournament approach. Compared to when all programmers receive equal rewards, the firm's profit is strictly lower when it uses the tournament approach if $\alpha > \sqrt{M^{*\upsilon}/\theta\upsilon}$ (proof given in the Appendix).

Further, given that the number of programmers hired by a closed source firm $M_{\rm CS}$ is likely to be less than the number of open source programmers $N_{\rm OS}$, a closed source firm that is not already competitive cannot compete with open source firms by internally adopting an open source strategy. The cost of hiring enough programmers to make the tournament approach work would be formidable.

It may however be possible for a proprietary firm to obtain some benefits by collaborating with the Open Source community by implementing a hybrid development system where the community is encouraged to develop add-on features or product extensions. For example, in the gaming software industry, it is common for the major players such as Sony to tap into the open source community to develop new features for a game while the core of the software is architected and developed in-house. A search of the Internet shows that a large number of software firms such as Adobe, Microsoft, and Sun offer software development kits (SDK) for free. An SDK consists of an interface, development tools, and libraries that assist programmers in creating application programs. However, the SDKs can be downloaded only after agreeing to a license whose terms may be unsuitable for free software development.

Licensing issues must be carefully considered so that the firm does not accidentally release its proprietary software as an open source. This is because code under the general public license (GPL) cannot be mixed with nonfree software. Alternative open source licenses where mixing is permitted include the Berkeley system distribution (BSD) license, the Mozilla public license, and the GNU Library GPL. Another approach is used by Sun, which has an Open Source office suite called http://www.openoffice.org and a commercial version called StarOffice. Beneficial contributions to http://www.openoffice.org can be separately implemented into StarOffice.

The firm must also remain vigilant about not making volunteer contributors feel exploited, which it will if it profits at the expense of their work [4]. This may cause them to lose altruism and stop contributing. Firms therefore offer rewards to contributors including putting them on the payroll.

An example of a hybrid model used by Microsoft is described by [38] and is called shared source.

> Microsoft is expanding its licensing model to include our *Shared Source philosophy*. Shared Source is a balanced approach that allows us to share source code with customers and partners while maintaining the intellectual property needed to support a strong software business... Over the years Microsoft has made millions of lines of source code freely available to developers through resources such as SDKs, DDKs, and MSDN... The commercial software model is just one model being utilized in the software industry today. It is important to take into account the Open Source Software movement as an example of an alternative model... the common traits are providing people with access to source code and allowing others to modify and redistribute that code. As a result of Microsoft's statement of position today, many people will attempt to say that Shared Source is Microsoft's failed attempt at being an Open Source Company. This could not be a more incorrect statement. Shared Source is not Open Source. The authors recognize that OSS has some benefits such as the fostering of community, improved feedback, and augmented debugging. The authors are always looking for ways to improve our products and make our customers more successful, and to that end the authors have incorporated these positive OSS elements in Shared Source.

So far, the authors have examined the improvement in quality under open source and closed source when each environment was modeled as a monopoly. The authors next examine quality implications in a competitive market where open source and closed source softwares compete for the same set of users.

VI. Competition Between Open and Closed Source

Let $Q_{\rm CS}$ and $Q_{\rm OS}$ be the qualities of the closed source and open source softwares, respectively, and P be the price of the closed source software. It is obvious that if $Q_{\rm CS} \leq Q_{\rm OS}$, then all D consumers will adopt the open source software. Competition arises only when $Q_{\rm CS} > Q_{\rm OS}$. (In the model here,

the demand for a software is a function of only its own and competitor's quality and price. Other factors such as industry standards and support may also influence a buyer's decision to adopt the open or closed source software. Such issues can be analyzed using location models such as linear or circular city model. The authors separately analyzed the competition using location models and found that the qualitative results of this section remain the same.) In this case, it can be observed easily that only consumers with higher valuations, i.e., higher t, will buy the closed source software and the rest will adopt the open source software. The indifferent consumer is the one whose surplus is the same for both software. That is, the indifferent consumer's t value is given by

$$tQ_{\mathrm{CS}}^{\theta} - P = tQ_{\mathrm{OS}}^{\theta} \Rightarrow t = \frac{P}{\left(Q_{\mathrm{CS}}^{\theta} - Q_{\mathrm{OS}}^{\theta}\right)}. \tag{14}$$

Thus, demands for the closed and open source softwares are given by $D(1 - P/(Q_{\mathrm{CS}}^{\theta} - Q_{\mathrm{OS}}^{\theta}))$ and $D\ P/(Q_{\mathrm{CS}}^{\theta} - Q_{\mathrm{OS}}^{\theta})$, respectively.

The authors assume the same sequence of actions described in the previous section on closed source under monopoly with the provision that under competition, in stage 2, open source and closed source qualities are simultaneously determined. As before, the closed source firm sets its price in stage 3.

For the closed source model, the authors obtain the quality as follow. The optimal price and profit are

$$P = \frac{\left(Q_{\mathrm{CS}}^{\theta} - Q_{\mathrm{OS}}^{\theta}\right)}{2} \tag{15}$$

$$\Pi_{\mathrm{CS}} = \frac{D\left(Q_{\mathrm{CS}}^{\theta} - Q_{\mathrm{OS}}^{\theta}\right)}{4}. \tag{16}$$

Using an analysis similar to that of the previous sections, it can be shown that

$$Q_{\mathrm{CS}}^{2-\theta}\left(Q_{\mathrm{CS}}^{\theta} - Q_{\mathrm{OS}}^{\theta}\right)^{1-\upsilon} = M_{\mathrm{CS}}\theta\upsilon\alpha^{2}\left[\frac{(1-\rho)D}{4}\right]^{\upsilon}\left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L}\right] \tag{17}$$

$$Q_{\mathrm{OS}} = \sqrt{\frac{N_{\mathrm{OS}}R\left(\frac{D}{2}\right)^{\upsilon}}{c}\left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L}\right]} \tag{18}$$

and welfare is

$$\frac{3D(Q_{\mathrm{OS}} + Q_{\mathrm{CS}})}{8}. \tag{19}$$

Hence, the following result holds.

Proposition 5: Under competition, 1) open source quality is independent of closed source quality; 2) open source quality is lower than under monopoly; 3) closed source profit, quality, and price are equal or lower than under monopoly, and are decreasing as open source quality increases; and 4) welfare increases (proof given in the Appendix).

Many of the results in Proposition 5 have important implications for the software market.

1) Open source quality is lower under competition than under monopoly. The reason for the lower quality in competition is that the closed source software takes away some of the market for the open source software, reducing the potential reward of open source programmers. Thus, open source programmers are likely to put in less effort.
2) Open source quality is independent of the closed source quality. The reason for this counter-intuitive finding is that the closed source firm, which can use price to manipulate demand, finds it optimal to charge a price such that it gets the demand from all and only consumers on the upper half of the valuation spectrum (i.e., consumers with $t > 1/2$). Because these consumers value quality more than consumers on the lower half of the valuation spectrum, they are more willing to pay a price for the higher quality closed source software. The open source firm caters to the consumers in the bottom half of the valuation spectrum. Because the open source firm does not manipulate demand through price (i.e., it charges a zero price for its product), the open source firm gets its demand from all and only consumers on the lower half of the valuation spectrum. This demand structure occurs irrespective of the closed source quality.
3) Closed source quality also decreases under competition from open source software. Conventional wisdom suggests that competition will increase quality and decrease price, as in the case of automobiles (Banker *et al.* 1998) and other industries. The authors find that for the software industry, while competition decreases price, it does not increase quality. The reason for this result is that the higher price charged due to higher quality is offset by the lower price (zero in the case of open source software) of the competing product. Faced with a decline in revenue, the closed source firm reduces its investment in quality.

VII. Conclusion

This paper examined software quality improvement under two distinct methods of software development: the traditional closed source approach (e.g., MS Windows) and the emerging open source approach (e.g., Linux). Software quality is conceptualized as improvements to the software package not only in terms of identifying and correcting errors but also enhancing the functionality, scalability, maintainability, portability, and, in general, enhancing the usability of the software. For both open and closed source softwares, quality was defined in this study in terms of acceptable enhancements to the code proposed by the programmer community.

Using game theoretic analysis and modeling open source and closed source software developments under monopoly and competitive settings, the authors derived several useful and interesting findings. They found that open source software quality is not necessarily lower than closed source quality despite the absence of a proprietary interest. Programmers are motivated to improve open source quality for an opportunity for recognition and future potential benefits.

This study demonstrated that there is no universal dominance of open source or closed source approaches in terms of software quality. Quality improvement under either scenario depends on exogenous variables such as market structure (monopoly

versus competition), incentive structure (tournament versus profit sharing), the number and quality of programmers, coordination between programmers, and endogenous variables such as reward, programmers' efforts, and number of users. It is virtually impossible to stake a claim that one approach is better than the other for all software and under all circumstances without simultaneous consideration of all of the above factors. In other words, there is no "one size fits all" approach to software development, and hence the authors believe that industry claims touting one approach over the other are fundamentally misguided. The adoption of Linux by large enterprises, e.g., Diamler–Chrysler, attests that they found it to be lower TCO for the same quality level. Yet other car manufacturers remain with Unix, a proprietary software. This is consistent with the prediction that there is no clear-cut dominance of Open Source over proprietary software or *vice versa*. Other than Linux, some data are available on the adoption of Apache, a web server software that competes with Microsoft. A recent survey at http://news.netcraft.com/archives/web_server_survey.html shows that about 70% of servers run on Apache and 20% on Microsoft.

However, it is still possible to compare marginal improvements in quality between open and closed source developments due to each of the above factors, when all else are 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 incentive to improve softwares 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). Second, the quality of open and closed source softwares also depends on the nature of the market (monopolistic versus competitive) and competitors in the market. The quality of both open source and closed source softwares 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*.

Finally, some critics believe that open source software quality suffers because programmers view code contributions as a hobby rather than as a serious pursuit or being paid for the same. Since many open source programmers come from a hacker culture, they perceive identifying bugs and fixes as a fun or enjoyable activity, and hence such hobby considerations are reasonable. The authors found that open source software quality decreases with programmers' cost of effort, and hobby considerations increases quality by reducing programmers' cost of effort. The enjoyment associated with open source code improvement motivates programmers to create and share improvements, even in the absence of pay.

Like most other analytical models, the current model suffers from a few limitations. First, it focused essentially on determining the equilibrium quality of open source software and not on the process of development over time. In other words, a dynamic model capturing features such as incremental quality improvements via peer feedback and collaboration on future entries to the tournament was not modeled. Restricting such indirect collaboration between programmers may have led to underestimation of the quality of open source software in the analysis. Second, the authors made certain simplifying assumptions such as quadratic cost functions. Since formal modeling relies on abstraction to address some issues in a stylized setting, limitations are inherent to the modeling process, but analysis of an abstract model can help understand, explain, or predict real issues in life. Future research may try to address some of the more restrictive assumptions.

Future research can extend the current analysis to a multiperiod game. Issues relevant in a multiperiod game such as determining per-period quality improvements, prices and upgrade prices, and the effect of customers' leap-frogging quality advances may be modeled. It was assumed in this paper that there is only one winner who receives the reward in the open source tournament. It is possible that there will be multiple winners with different levels of rewards in this tournament. More complex interdependence of programmers' efforts in determining the total software quality, as opposed to the summation of qualities used here, could also be examined. Other issues such as standards, licensing policies, and legal issues remain to be investigated in the open source context. The model approach is necessarily an approximation of actual approaches from the industry. It would also be fruitful to continue to study approaches taken by industry on quality development in open source. This might include case comparisons, cross-sectional analysis, and longitudinal evolution of quality under the two methods.

APPENDIX

Proof of Proposition 1

Consider the Nth programmer and apply a statistical theorem [40, p. 183] as an intermediate step.

Theorem 1: If $\{Q_1, Q_2, \ldots, Q_{N-1}\}$ are independent random variables with cdfs $\{F_{Q_1}, F_{Q_2}, \ldots, F_{Q,N-1}\}$, respectively, and $Y_{N-1} \equiv \max\{Q_1, Q_2, \ldots, Q_{N-1}\}$, then the cdf of Y_{N-1} is $F_{Y_{N-1}} = \prod_{j=1}^{N-1} F_{Qj}$.

Proof of Theorem 1: $\mathrm{Prob}(Y_{N-1} < Q) = \mathrm{Prob}(Q_1 < Q) \cap \mathrm{Prob}(Q_2 < Q) \cap \cdots \cap \mathrm{Prob}(Q_{N-1} < Q) = \prod_{j=1}^{N-1} F_{Qj}$.

Note that $p(q_i, q_{-i}) = \mathrm{prob}(Q_i - Y_{N-1} \geq 0)$. In other words $p(\cdot)$ is the cdf of the random variable $(Y_{N-1} - Q_i)$ and can be determined using the convolution theorem [40, p. 186]

$$p(Y_{N-1} - Q_i \leq 0) = \int_{-\infty}^{\infty} f_{Q_i}(x) \prod_{j=1}^{N-1} F_{Q_j}(x) \mathrm{d}x.$$

If $Y_N \equiv \prod_{j=1}^{N} F_{Qj}$ is the distribution of the maximum quality, its expectation is obtained by the formula $E(Y_N) = \int_0^{\infty}(1 - F_{Y_N}) - \int_{-\infty}^{0} F_{Y_N}$ [40, p. 65]. ■

Proof of Corollary 1.1

Starting with (1) and the tournament winner assumption

$$U_i = \frac{q_i}{\sum_{j=1}^{N_{\mathrm{OS}}} q_j} R(D_{\mathrm{OS}})^{\upsilon} - C_{\mathrm{OS}}(q_i). \tag{A1}$$

Solving for the quality chosen, the first-order condition is

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

$$\Rightarrow \frac{R(D_{\rm OS})^{\upsilon}\left[(N_H-1)q_H + N_L q_L\right]}{[N_H q_H + N_L q_L]^2} - c\lambda_H q_H = 0$$

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

Further, this implies

$$\frac{R(D_{\rm OS})^{\upsilon}}{(N_H q_H + N_L q_L)c\lambda_H} \approx q_H$$

$$\text{and } \frac{R(D_{\rm OS})^{\upsilon}}{(N_H q_H + N_L q_L)c\lambda_L} \approx q_L \tag{A2}$$

where the symbol $\approx$ denotes "approximated by." 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, the authors get $\lambda_L/\lambda_H = q_H/q_L$. Multiplying the first equation by N_H and the second by N_L and adding, the authors get the expected quality of the software

$$Q_{\rm OS} = \sqrt{R(D_{\rm OS})^{\upsilon}\left(\frac{N_H}{c\lambda_H} + \frac{N_L}{c\lambda_L}\right)}$$

$$= \sqrt{\frac{N_{\rm OS}R(D)^{\upsilon}}{c}\left(\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L}\right)}. \tag{A3}$$

Since there is no firm profit, social welfare is the surplus of the consumers. This is given by

$$D\int_0^1 tQ_{\rm OS}^{\theta}\,{\rm d}t = \frac{DQ_{\rm OS}^{\theta}}{2}. \tag{A4}$$

■

Proof of Proposition 2

Proof by contradiction: suppose a separating contract $(x_1, Q_{\rm CS})$ and $(x_2, Q_{\rm CS})$ is announced where $x_1 > x_2$ are the proportions of profit shared. These must satisfy the incentive compatibility constraints

$$\text{IC}_1: \quad \Pi_1(x_1, Q_{\rm CS}|\, x_1) > \Pi_1(x_2, Q_{\rm CS}|\, x_1)$$

$$\text{IC}_2: \quad \Pi_2(x_2, Q_{\rm CS}|\, x_2) > \Pi_2(x_1, Q_{\rm CS}|\, x_2).$$

Clearly, 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 a low-cost programmer without any cost incurred for lying. ■

Proof of Proposition 3

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

$$\Pi_{\rm CS}(Q_{\rm CS}) = \operatorname*{Max}_P D\left(1 - \frac{P}{Q_{\rm CS}^{\theta}}\right)P. \tag{A5}$$

The optimal price is $Q_{\rm CS}^{\theta}/2$. Inserting this into the profit expression gains

$$\Pi_{\rm CS}(Q_{\rm CS}) = \frac{DQ_{\rm CS}^{\theta}}{4}. \tag{A6}$$

Substituting this into the Stage 2 analysis gives the utility of the programmer

$$\left[\frac{(1-\rho)DQ_{\rm CS}^{\theta}}{4M_{\rm CS}}\right]^{\upsilon} - \lambda_i\frac{q_i^2}{2}. \tag{A7}$$

The first-order condition can be rewritten for the two programmer types separately as

$$\frac{\alpha\theta\upsilon}{\lambda_L}\left[\frac{(1-\rho)D}{4M_{\rm CS}}\right]^{\upsilon}(Q_{\rm CS})^{\theta\upsilon-1} = q_L \tag{A8}$$

$$\frac{\alpha\theta\upsilon}{\lambda_H}\left[\frac{(1-\rho)D}{4M_{\rm CS}}\right]^{\upsilon}(Q_{\rm CS})^{\theta\upsilon-1} = q_H. \tag{A9}$$

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

$$\frac{q_H}{q_L} = \frac{\lambda_L}{\lambda_H}. \tag{A10}$$

This implies that the low-cost programmers choose a higher quality level than the high-cost programmers. The authors now solve for software quality. Multiplying the first equation by αM_H and the second by αM_L and adding, the authors solve as

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

$$\Rightarrow Q_{\rm CS} = \left(M_{\rm CS}^{1-\upsilon}\theta\upsilon\alpha^2\left[\frac{(1-\rho)D}{4}\right]^{\upsilon}\left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L}\right]\right)^{\frac{1}{2-\theta\upsilon}}. \tag{A11}$$

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

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

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

$$M_{\rm CS} = \left(\frac{D^2\alpha^{2\theta}\theta^3\upsilon^{\theta(\upsilon+1)}(1-\upsilon)^{2-\theta\upsilon}\left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L}\right]^{\theta}}{64k^{2-\theta\upsilon}}\right)^{\frac{1}{2-\theta}}. \tag{A12}$$

Substituting these back into the quality expression obtains

$$Q_{\mathrm{CS}} = \left(\frac{D^{\frac{2}{\theta}} \alpha^2 \theta^{1+\frac{1}{\theta}} \upsilon^{\upsilon+1} (1-\upsilon)^{1-\upsilon} \left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L} \right]}{8k^{1-\upsilon}} \right)^{\frac{1}{2-\theta}}. \tag{A13}$$

Social welfare is the sum of the firm's profits and consumer surplus. Since half the market is served by the firm, profit is given by $DP/2 = DQ_{\mathrm{CS}}^{\theta}/4$. Surplus is

$$D \int_{\frac{1}{2}}^{1} \left(t Q_{\mathrm{OS}}^{\theta} - P \right) \mathrm{d}t = \frac{D Q_{\mathrm{OS}}^{\theta}}{8}. \tag{A14}$$

Hence, welfare is equal to $(DQ_{\mathrm{CS}}^{\theta}/4) + (DQ_{\mathrm{CS}}^{\theta}/8) = 3DQ_{\mathrm{CS}}^{\theta}/8$. ■

Proof of Proposition 4

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

Proof of Corollary 4.1

If the closed source firm uses the tournament approach, then the reward function for the programmers is given by $(1-\rho) DQ_{\mathrm{CS}}^{\theta}/4$. In the tournament approach, $\alpha = 1$ as in the open source. From (1) and tournament winner assumption, utility is

$$U_i = \frac{q_i}{\sum_{j=1}^{N_{\mathrm{OS}}} q_j} \left(\frac{(1-\rho) D Q_{\mathrm{CS}}^{\theta}}{4} \right)^{\upsilon} - C_{\mathrm{OS}}(q_i). \tag{A15}$$

Solving for planned quality, the first-order condition is

$$\left(\frac{(1-\rho)D}{4} \right)^{\upsilon} \left(Q^{\theta\upsilon-1} + (\theta\upsilon - 1) Q^{\theta\upsilon-2} q_i \right) = \lambda_i q_i. \tag{A16}$$

Assuming that the quality of a single programmer is small compared to the total quality, the authors get the first-order condition

$$\left(\frac{(1-\rho)D}{4} \right)^{\upsilon} \frac{Q^{\theta\upsilon-1}}{\lambda_i} = q_i. \tag{A17}$$

Multiplying the equation for high-cost programmers by M_H and for the low cost by M_L and adding, the authors get

$$M_{\mathrm{CS}} \left[\frac{(1-\rho)D}{4} \right]^{\upsilon} \left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L} \right] = (Q_{\mathrm{CS}})^{2-\theta\upsilon}. \tag{A18}$$

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

$$\underset{\rho, M_{\mathrm{CS}}}{\mathrm{Max}} \frac{\rho D \left(M_{\mathrm{CS}} \left[\frac{(1-\rho)D}{4} \right]^{\upsilon} \left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L} \right] \right)^{\frac{\theta}{2-\theta\upsilon}}}{4} - k M_{\mathrm{CS}}. \tag{A19}$$

Let M^* be the optimal value of M_{CS}. If the closed source firm does not use the tournament approach, its optimization model is given by

$$\underset{\rho, M_{\mathrm{CS}}}{\mathrm{Max}} \frac{\rho D \left(M_{\mathrm{CS}}^{1-\upsilon} \theta\upsilon\alpha^2 \left[\frac{(1-\rho)D}{4} \right]^{\upsilon} \left[\frac{\beta}{\lambda_H} + \frac{1-\beta}{\lambda_L} \right] \right)^{\frac{\theta}{2-\theta\upsilon}}}{4} - k M_{\mathrm{CS}}. \tag{A20}$$

Comparing (A19) and (A20), the authors can note that for the closed source firm to be worse off using the tournament approach, $\theta\upsilon\alpha^2 > M^{*\upsilon}$ is a sufficient condition. ■

Proof of Proposition 5

1) The result is obvious from the Q_{OS} expression.
2) By comparing the expressions for Q_{OS} under monopoly and competition, and noting that R is an increasing function, the authors show this result.
3) The authors prove this result by contradiction. They know that when $Q_{\mathrm{OS}} = 0$, Q_{CS} is same under competition and monopoly. Let us assume a positive Q_{OS}. Let the optimal M, ρ, and Q_{CS} for the competition case be M_C, ρ_C, and Q_{CS}^C, respectively. Let the optimal M, ρ, and Q_{CS} for the monopoly case be M_M, ρ_M, and Q_{CS}^M, respectively.

Note that $\partial Q_{\mathrm{CS}}/\partial Q_{\mathrm{OS}} \leq 0$ under competition. Thus, the closed source can obtain strictly higher profit with M and ρ values equal to M_C and ρ_C, respectively, in the monopoly case because for these values of M and ρ, costs will be identical in the monopoly and competition cases but Q_{CS}, and consequently price and demand, will be higher in the monopoly case. Hence, the optimal profit in the monopoly case cannot be lower than that in the competition case.

Assume that $Q_{\mathrm{CS}}^C > Q_{\mathrm{CS}}^M$. If the values of M and ρ in the monopoly case are set to M_{C} and ρ_{C}, respectively, then Q_{CS} will be higher than Q_{CS}^M. Consequently, under monopoly, for these values of M_C and ρ_C, the profit will be higher than the optimal profit, which contradicts the optimality of M_M and ρ_M.

The result that price is lower under competition follows from the expression for price and the results that Q_{CS} is lower under competition.

Welfare is given by the following expression, where the three terms, respectively, are the surplus of open source software users, the profit for the firm, and the surplus of closed source software users:

$$\begin{aligned} & D \int_{0}^{\frac{1}{2}} t Q_{\mathrm{OS}}^{\theta} \mathrm{d}t + \frac{D Q_{\mathrm{CS}}^{\theta}}{4} + D \int_{\frac{1}{2}}^{1} (t Q_{\mathrm{CS}}^{\theta} - P) \mathrm{d}t \\ &= \frac{D Q_{\mathrm{OS}}^{\theta}}{8} + \frac{D Q_{\mathrm{CS}}^{\theta}}{4} + \frac{D Q_{\mathrm{OS}}^{\theta}}{4} + \frac{D Q_{\mathrm{CS}}^{\theta}}{8} \\ &= \frac{3D(Q_{\mathrm{OS}}^{\theta} + Q_{\mathrm{CS}}^{\theta})}{8}. \end{aligned} \tag{A21}$$

■

ACKNOWLEDGMENT

The authors thank A. Bhattacharjee, D. Harter, E. Haruvy, and seminar participants at the University of Texas at Dallas and the 23rd International Conference on Information Systems (ICIS) in Barcelona 2002.

REFERENCES

[1] T. O'Reilly, "Lessons from open-source software development," *Commun. ACM*, vol. 42, no. 4, pp. 32–37, 1999.
[2] G. Von Krogh, "Open source software development," *Sloan Manage. Rev.*, vol. 44, no. 3, pp. 14–18, 2003.
[3] E. S. Raymond. (1999). The Cathedral and the Bazaar. [Online]. Available: http://www.tuxedo.org/~esr/writings/magic-cauldron/magic-cauldron.txt
[4] E. Haruvy, A. Prasad, and S. P. Sethi, "Harvesting altruism in open source software development," *J. Optim. Theory Appl.*, vol. 118, no. 2, pp. 381–416, 2003.
[5] G. P. Dwyer, Jr., "The economics of open source and free software," Federal Reserve Bank of Atlanta, 1999, unpublished paper.
[6] I.-L. Hann, J. Roberts, S. Slaughter, and R. Fielding, "Economic incentives for participating in open source software projects," in *Proc. 23rd Int. Conf. Information Systems*, Barcelona, Spain, 2002, pp. 365–372.
[7] J. Lerner and J. Tirole, "Some simple economics of open source," *J. Ind. Econ.*, vol. 52, no. 2, pp. 197–234, 2002.
[8] Y. Benkler, "Coase's penguin, or Linux and the nature of the firm," *Yale Law J.*, vol. 112, no. 3, pp. 369–389, 2002.
[9] E. Von Hippel and G. Von Krogh, "Open source software and the 'private-collective' innovation model: Issues for organizational science," *Organ. Sci.*, vol. 14, no. 2, pp. 209–313, 2003.
[10] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proc. 2nd Int. Conf. Software Engineering*, San Francisco, CA, 1976, pp. 592–605.
[11] I. Gorton and A. Liu, "Software component quality assessment in practice: Successes and practical impediments," in *Proc. 24th Int. Conf. Software Engineering*, Orlando, FL, 2002, pp. 555–558.
[12] D. E. Harter, M. S. Krishnan, and S. A. Slaughter, "Effects of process maturity on quality, cycle time, and effort in software product development," *Manage. Sci.*, vol. 46, no. 4, pp. 451–466, 2000.
[13] C. C. Mann. (2002). "Why software is so bad... and what's being done to fix it." *Tech. Rev.* [Online]. Available: www.technologyreview.com/articles/mann0702.asp
[14] C. K. Prahalad and M. S. Krishnan, "The new meaning of quality in the information age," *Harvard Bus. Rev.*, vol. 77, no. 5, pp. 109–118, 1999.
[15] S. Slaughter, D. Harter, and M. Krishnan, "Evaluating the cost of software quality," *Commun. ACM*, vol. 41, no. 8, pp. 67–73, 1998.
[16] Bloor Research. (1999, Oct. 22). Linux Versus NT: The Verdict. [Online]. Available: http://www.itsecurity.com/news4/p222.htm
[17] C. Laird. (1998, Aug.). Linux Versus NT: Are You Getting the Most From Your OS? [Online]. Available: http://www.sunworld.com/sunworldonline/swol-08-1998/swol-08-linuxvnt.html
[18] IDC. (2002). Windows 2000 Versus Linux in Enterprise Computing. [Online]. Available: http://www.microsoft.com/windows2000/docs/TCO.pdf
[19] Robert Frances Group. (2002). Total Cost of Ownership for Linux in the Enterprise. [Online]. Available:http://www.ibm.com/linux/RFG-LinuxTCO-vFINAL-Jul2002.pdf
[20] J. R. Green and N. L. Stokey, "A comparison of tournaments and contracts," *J. Polit. Econ.*, vol. 91, no. 3, pp. 349–365, Jun. 1983.
[21] S. Rosen, "Prizes and incentives in elimination tournaments," *Amer. Econ. Rev.*, vol. 76, no. 4, pp. 701–715, 1986.
[22] L. Li and Y. S. Lee, "Pricing and delivery-time performance in a competitive environment," *Manage. Sci.*, vol. 40, no. 5, pp. 633–646, 1994.
[23] P. J. Lederer and L. Li, "Pricing, production, scheduling, and delivery-time competition," *Oper. Res.*, vol. 45, no. 3, pp. 407–420, 1997.
[24] K. C. So, "Price and time competition for service delivery," *Manuf. Serv. Oper. Manag.*, vol. 2, no. 4, pp. 392–409, 2000.
[25] A. Tsay and N. Agrawal, "Channel dynamics under price and service competition," *Manuf. Serv. Oper. Manag.*, vol. 2, no. 4, pp. 372–391, 2000.
[26] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: The Apache server," in *Proc. 22nd Int. Conf. Software Eng.*, Limerick, Ireland, 2000, pp. 263–272.
[27] K. Crowston and B. Scozzi, "Open source software projects as virtual organizations: Competency rallying for software development," *Proc. Inst. Elect. Eng., Softw.*, vol. 149, no. 1, pp. 3–17, 2002.
[28] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory*. New York: Oxford Univ. Press, 1995.
[29] A. K. Basu, R. Lal, V. Srinivasan, and R. Staelin, "Salesforce compensation plans: An agency theoretic perspective," *Mark. Sci.*, vol. 4, no. 4, pp. 267–291, 1985.
[30] D. McFadden, "Conditional logit analysis of qualitative choice behavior," in *Frontiers in Econometrics*, P. Zarembka, Ed. New York: Academic, 1974, pp. 105–142.
[31] G. Loury, "Market structure and innovation," *Q. J. Econ.*, vol. 93, no. 3, pp. 395–410, 1979.
[32] R. D. Luce, *Individual Choice Behavior*. New York: Wiley, 1959.
[33] L. G. Cooper and M. Nakanishi, *Market Structure Analysis*. Norwell, MA: Kluwer, 1998.
[34] S. J. Grossman and O. D. Hart, "An analysis of the principal-agent problem," *Econometrica*, vol. 51, no. 1, pp. 7–46, 1983.
[35] B. Holmstrom, "Moral hazard in teams," *Bell J. Econ.*, vol. 13, no. 2, pp. 324–340, 1982.
[36] H. Itoh, "Incentives to help in multi-agent situations," *Econometrica*, vol. 59, no. 3, pp. 611–636, 1991.
[37] B. E. Hermalin, "Towards an economic theory of leadership: Leading by example," *Amer. Econ. Rev.*, vol. 88, no. 5, pp. 1188–1205, 1998.
[38] C. Mundie. (2001, May 3). The Commercial Software Model. [Online]. Available: http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp
[39] R. D. Banker, I. Khosla, and K. K. Sinha, "Quality and competition," *Manage. Sci*, vol. 44, no. 9, pp. 1179–1192, 1998.
[40] A. Mood, F. A. Graybill, and D. C. Boes, *Introduction to the Theory of Statistics*, 3rd ed. Singapore: McGraw-Hill, 1974.

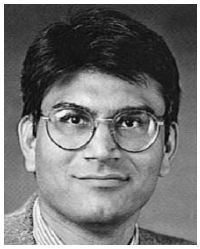
Srinivasan Raghunathan received the M.S. degree in business administration from the Indian Institute of Management (IIM) Calcutta, West Bengal, India, and the Ph.D. degree from the University of Pittsburgh, Pittsburgh, PA.

He is an Associate Professor in the School of Management, University of Texas at Dallas, Richardson, TX. His research interests include economics of information systems, IT security, and information sharing in supply chains. His research is published in several journals including Management Science, Information Systems Research, and various IEEE TRANSACTIONS.

Ashutosh Prasad received the M.S. degree in economics and the Ph.D. degree in marketing from the University of Texas at Austin, Austin, TX, and the M.S. degree in business administration from the Indian Institute of Management (IIM) Calcutta, West Bengal, India.

He is an Assistant Professor in the School of Management, University of Texas at Dallas, Richardson, TX. His research is published or forthcoming in several journals including the *International Journal of Research in Marketing*, *Journal of Business*, *Journal of Optimization Theory and Applications*, *Marketing Letters*, and *Marketing Science*. His research interests include new product introduction, network effects, software marketing and piracy, advertising, and sales force issues.

Dr. Prasad received the School of Management's Outstanding Undergraduate Teacher Award in 2001.



Birendra K. Mishra received the M.S. degree in petroleum engineering and the Ph.D. degree in accounting from the University of Texas at Austin, Austin, TX.

He is a Visiting Assistant Professor at the Anderson Graduate School of Management, University of California, Riverside, CA, and an Assistant Professor at the School of Management, University of Texas at Dallas, Richardson, TX. He uses agency theory and game theory to model and analyze business problems. His research has appeared or is forthcoming in *Marketing Science*, *Management Science*, *Journal of Accounting Research*, *Journal of Accounting and Public Policy*, *Journal of Management Accounting Research*, and *Journal of Business Research* among others.

Dr. Mishra was awarded the Dean's Excellence in Research Grant at the Arizona State University.

Hsihui Chang received the Ph.D. degree in accounting from the University of Minnesota, Minneapolis, MN.

He is an Associate Professor of Accounting and the Associate Dean at the Gary Anderson Graduate School of Management, University of California, Riverside, CA. His current research interests include performance measurement, information systems, and data envelopment analysis. He has published more than 25 articles in various leading research journals including the *European Journal of Operational Research*, *Journal of Accounting and Economics*, *Journal of Accounting and Public Policy*, *Strategic Management Journal*, and *The Accounting Review*.

Dr. Chang has received many awards for his research articles and one award for his outstanding teaching. He has supervised an award-winning doctoral dissertation. He is an active member of various professional organizations and has served as a consultant to several profit and nonprofit organizations.