MANAGEMENT SCIENCE
Vol. 52, No. 7, July 2006, pp. 1000–1014
ISSN 0025-1909 | EISSN 1526-5501 | 06 | 5207 | 1000

informs

DOI 10.1287/mnsc.1060.0553

Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development

Sonali K. Shah
University of Illinois at Urbana–Champaign, 465 Wohlers Hall, Champaign, Illinois 61820, sonali@uiuc.edu

Open source software projects rely on the voluntary efforts of thousands of software developers, yet we know little about why developers choose to participate in this collective development process. This paper inductively derives a framework for understanding participation from the perspective of the individual software developer based on data from two software communities with different governance structures.

In both communities, a need for software-related improvements drives initial participation. The majority of participants leave the community once their needs are met, however, a small subset remains involved. For this set of developers, motives evolve over time and participation becomes a hobby. These hobbyists are critical to the long-term viability of the software code: They take on tasks that might otherwise go undone and work to maintain the simplicity and modularity of the code. Governance structures affect this evolution of motives. Implications for firms interested in implementing hybrid strategies designed to combine the advantages of open source software development with proprietary ownership and control are discussed.

Key words: open source software development; innovation; motivation; volunteers; governance
History: Accepted by Eric von Hippel and Georg von Krogh, guest editors; received September 1, 2004. This paper was with the author 5 months and 3 weeks for 3 revisions.

1. Introduction

> I pick and choose the work that's most interesting to me . . . . It's great when you find a challenging problem to work on—either on your own or because someone needs it—you can spend hours on it . . . . (Long term open source community participant, United States)

> In an open source community, not one answer is forced on anyone. Everything is up for discussion and change—all the time. Sometimes it gives me a headache [chuckle] . . . it's empowering and it leaves room for new people to come in and make improvements and changes . . . . That dynamic just doesn't exist in communities around tightly licensed corporate code—or in the companies that most of us work for. (Expert, United States)

Open source software projects such as Linux, Apache, and Gnome have achieved remarkable success. These projects provide participants with the social context and resources to create useful and publicly available software that has, on occasion, displaced commercially produced software. The projects are distinctive in that they rely on the efforts of a community of volunteer software users and developers instead of paid managers and employees. These open source software projects are thus exemplars of a fundamentally different organizational model for innovation and product development—referred to as collective invention (Allen 1983), private-collective innovation (von Hippel and von Krogh 2003), and community-based innovation (Franke and Shah 2003, Shah 2005). This model extends well beyond the domain of software: innovative communities have been influential in fields as diverse as astronomy (Ferris 2002), automobiles (Franz 2005, Kline and Pinch 1996), sports equipment (Franke and Shah 2003, Shah 2000), personal computers (Freiberger and Swaine 2000), and video games (Jeppesen and Molin 2003).

This paper investigates two specific research questions from an empirical perspective. First, why do individuals participate in innovation communities? And, second, how do differences in governance affect individuals' reasons for participation, as well as the type and quality of their contributions? To address these questions, this paper examines the motives of participants in two large and well-known community-based software development projects. One community is *open source* and the other is *gated source*, meaning that the communities rely on different governance structures, although both seek to attract the efforts of volunteer software developers. The study of two communities with different governance structures allows me to investigate how such differences affect individual participation. Data collected through interviews and publicly available project information are analyzed using an inductive and qualitative methodological approach based on the principles of grounded

theory building. This approach avoids layering preconceived theoretical concepts onto a novel social structure and "makes room for the discovery of the unanticipated (Van Maanan 1998)."

Three key findings were uncovered. First, reasons for participation vary, but a critical subset of open source developers—hobbyists—participate because they derive enjoyment from the act of creating code. Second, reciprocity is a crucial factor in motivating contributions to the community. Third, governance structure has a first-order effect on patterns of participation. Two additional distinctions not present in the existing literature also emerged. First, the creation and contribution of code and knowledge to the community are two different activities and are motivated by different sets of factors.[1] Second, the outcomes that motivate behavior are not always the same as the outcomes generated by the behavior. For example, some developers do attain status within communities and a select few have leveraged their open source efforts into high-paying jobs. Yet, it does not appear that these developers originally participated in the hopes of obtaining these rewards.

In the next section, I summarize previous research on the motives of open source software developers. I provide a detailed description of the study setting and method in §3. Findings are presented in §4. Implications of these findings for the governance of community-based projects are discussed in §5. Section 6 concludes.

2. Why Do Individuals Participate in Innovation Communities?

We are accustomed to thinking of firms, independent inventors, and research institutions as the primary engines of innovative activity and industrial progress. In theory, the research and development efforts of most firms and independent inventors are based on a proprietary benefit model. In this model, exclusive property rights are the basis for capturing value from innovative investments. Firms and independent inventors strive to innovate in the hopes of realizing profits from products protected by patents, copyrights, and/or trade secrets.

The research and development activities of universities and research institutions are based on an academic model. Publication, status, and prestige are the rewards for innovative activity and full-time, professionally trained and compensated researchers direct their own efforts toward these goals, under the aegis of funding agencies, such as the National Science Foundation and National Institutes of Health.

The proprietary and academic models do not, however, stand alone. Another model—"community-based" innovation—has generated innovations that we use on a daily basis. In contrast to the proprietary and academic models, the community-based model relies neither on exclusive rights nor hierarchical control. The model is instead based on the open, voluntary, and collaborative efforts of users—a term that describes enthusiasts, tinkerers, amateurs, everyday people, and even firms that derive benefit from a product or service by using it. Open source software development is a prominent example of the community-based model.[2]

Researchers studying open source software development are interested in the question of participation or, as Lerner and Tirole (2002) so aptly ask: Why are large numbers of talented developers voluntarily contributing to the creation, maintenance, and support of a public good? A number of competing and contentious theories have been suggested. Stallman (2001) argues that the free software ideology fuels open source software development. Others argue that participation is driven by users' desire to satisfy their own needs (Franke and von Hippel 2003, Kuan 2001, Lakhani and von Hippel 2003, Raymond 1999); career concerns, learning, and reputation (Hann et al. 2002, Lerner and Tirole 2002); reputation and status within the community (Raymond 1999); affiliation and identity (Hertel et al. 2003); or enjoyment and creativity (Gelernter 1998, Ghosh 1998, Weizenbaum 1976). By and large, each of these motives has been studied independently and some evidence has been found in support of each. Cross-sectional survey-based studies examine several motives and find evidence in support of each (Ghosh et al. 2002, Hertel et al. 2003). However, research has yet to devise a coherent explanation for these findings, connect these motives to the social structure, and understand how differences in the social structure affect participation and vice versa. The grounded theory-based approach employed in this study will help illuminate these patterns.

[1] Participants contribute a great many things to the community, including both knowledge and code. However, for ease of exposition, I will simply use the phrase "create and contribute code" throughout the remainder of the paper.

[2] One might also view innovation communities as volunteer organizations. Formally defined, volunteering is any activity in which time is given freely to benefit another person, group, or cause. This definition does not however preclude volunteers from benefiting from their work, although there is debate over whether work is truly voluntary if it is remunerated (Wilson 2000). From this perspective, participation in an open source community might be seen as an activity that benefits both the community and the participant (this is in line with descriptions of open source development as a private-collective innovation model, see von Hippel and von Krogh 2003). Evidence supporting a variety of theories that seek to explain volunteering from the perspective of individual attributes (emphasizing either cost-benefit analysis or identity) as well as social resources has been found. Less attention has been paid to contextual effects (i.e., the impact of the organization, community, and region) on an individual's decision to volunteer and this is considered a fruitful area for research (Wilson 2000).

In this study, I examine participation in considerable depth and consider the full set of possible motives, rather than seek to confirm or disprove any single reason for participation. An individual may possess multiple motives, motives may differ across individuals, and motives may evolve over time (Jencks et al. 1988, Jensen and Meckling 1994). These possibilities are recognized in the design of the study. The study also pays attention to individuals' perceptions of the community as a social system because each developer's choices are likely to depend on the surrounding social system and to affect that system as well (Coleman 1994, Giddens 1984). Therefore the goals, norms and values, status markers, and social cleavages of both communities were also recorded.

3. Research Method

I gathered data from multiple sources within two distinct software communities. In all, 88 interviews, over 2,000 mailing list postings, and online project documentation informed this research. The individual software developer is the focal unit of analysis. The study setting, data sources, and data analysis procedures are discussed in detail below.

Study Setting

As of October 2005, more than 105,000 open source projects exist (SourceForge 2005).[3] Not surprisingly, a wide variety of social structures and licensing structures also exist. I distinguish between two *general* approaches to community-based software development: open and gated. Both seek to attract volunteer developers, although different governance principles apply. To be considered "open source," a project's licensing terms must meet the 10 requirements of the open source definition (Open Source Initiative 2004). Gated approaches attempt to selectively combine the benefits of collective development with the corporate benefits of private ownership and control. They are more restrictive than open source licenses from the perspective of the developer. I chose to study participation in both types of communities in order to examine whether and how differences in governance structures affect the type, quality, and reasons for participation.

Two communities were identified based on interviews with six open source software experts and Internet searches. Both have a large number of participants and are well known within the software-development community. Both focus on the development of innovative software, that is, software that possesses a unique software architecture and creates novel functionality not found in existing software programs. They are of roughly the same age and use the same programming language. Both serve as umbrella organizations for multiple projects. Representative projects within each community were chosen based on conversations with four experts within the communities.

The key distinctions between the open and gated communities are as follows. In the open source community, anyone can download, use, modify, and distribute the code. The code is owned by the collective and a special subset of developers, called committers, settle contested project decisions. In the gated source community, only those who have agreed to a license with the corporate sponsor can download, use, or modify the code. The license stipulates that the code may only be shared with other licensees. The corporate sponsor owns the code and retains the right to make project decisions. Finally, licensees who use the code for commercial purposes must pay a royalty to the corporate sponsor.

Data Sources

Multiple sources of data are critical to grounded theory development as they enable triangulation and validation of theoretical constructs. Data from three primary sources informed this research (Table 1). All data were collected in 2001–2002.

Online Project Documentation. Online archives contained project descriptions, charters, bylaws, meeting minutes, and an informal survey of open source committers. This documentation provided a basis for understanding the formal rules and structure of the communities.

Table 1 Data Sources

Data source	Description
Online project documentation	All publicly available project descriptions, charters, bylaws, meeting minutes, etc.
Mailing lists	Over 2,000 messages read over a three-month period prior to interviews
Interviews	88 interviews conducted
Theoretically sampled interviews	45 interviews, selected to maximize variance
Snowball-sampled interviews	19 interviews, selected based on recommendations of previous interviewees
Gated source conference interviews	13 interviews, chosen opportunistically
Interviews with employees of gated source sponsor	11 interviews, selected based on employment

[3] Very few of these projects are able to construct functioning software or robust, if small, communities: only 75% of projects hosted on the SourceForge.net website contain code in their code repositories and 95% of projects have five or fewer contributors (Healey and Schussman 2003).

Mailing Lists. I read all postings to project-specific and general mailing lists for both communities for a three-month period preceding the interviews. Over 2,000 messages were posted during this period. Reading these messages allowed me to gain familiarity with the technology; the types, quantity, and content of discussions; and the contributions and roles of different individuals.[4]

A list of project participants was assembled from the mailing list and notes were made describing each participant's characteristics and activities. From these notes, I constructed a set of observable dimensions on which participants actions and behaviors differed. These dimensions became the backbone of the theoretical sampling strategy used to select over half of all interviewees.

Interviews. In total, 88 interviews were conducted. Interviewees were selected through the use of four complementary sampling strategies, which are described below. This choice of complementary strategies is an intentional part of the study design, used to ensure that the diversity of the community was captured in the data collection process.

Theoretically Sampled Interviews. I conducted 45 theoretically sampled interviews with project volunteers. Interviewees were chosen to maximize variance on the following four dimensions:

(a) Length of participation. The length of time since an individual's first post to the project mailing list. A *short-term participant* posted their first message less than two months prior to the interview. A *long-term participant* posted their first message to the mailing list more than two months prior to the interview.

(b) Current frequency of participation. The number of posts made to mailing lists in the preceding one-month time period.

(c) Type(s) of contributions made to mailing lists. Did the participant pose questions, provide answers or suggestions, make bug fixes, contribute code, participate in general discussions, or engage in a combination of the above activities?

(d) Individual's role within the community. Was the participant a user, developer, or a committer? A committer is a participant who has been granted access to the source code repository (this is only applicable to participants of the open source community).[5]

Information on these dimensions was gathered from observed behavior on project mailing lists (a, b, c) or other project-related documentation (d). The sample of 45 referred to in the tables includes only theoretically-sampled interviews.

Snowball-Sampled Interviews. I conducted 16 snowball-sampled interviews with members of the open source community and 3 with members of the gated community. Snowball sampling identifies interviewees based on the recommendations of past interviewees (Denzin and Lincoln 2000). In the course of this research, informants often recommended others who they viewed as knowledgeable, critical to the project, or having a unique perspective. I took the opportunity to speak to many such individuals in order to increase variance in the data collected.

Conference Interviews. Over 28 hours in a three-day period were spent observing and meeting with attendees at a technical conference focused on the gated source project. Here 13 interviews were conducted and a number of short (5–10 minute) conversations were also initiated. Conversations were not recorded because of the nature of interaction at such conferences; detailed notes were taken at the end of the day and, when appropriate, during conversations.

Interviews with Employees of the Corporate Sponsor. I conducted interviews with 11 employees of the gated project's corporate sponsor. They represented various functional roles within the firm and were employed at levels throughout the firm's hierarchy. Interview questions focused on the nature of work undertaken by employees of the sponsor, project history, corporate priorities, complementarities and tensions between corporate and community priorities, and interactions with volunteer participants.

In the formal interviews, subjects were asked a series of open-ended questions, augmented by follow-up and clarifying questions (Spradley 1979). Questions addressed the following issues: (1) the participants' initial and current usage of the software, (2) project-related activity, (3) methods for processing mailing list information, (4) background, (5) current employment, and (6) perceptions of the governance practices used within the project. Interviewees were guaranteed anonymity to promote candid responses.

As part of the study design, I did not inquire about motives directly. Doing so leads subjects to rationalize their actions (Becker 1998; Spradley 1979, pp. 81–82). Instead, I asked concrete questions about the subject's

[4] Many people download and use the software but never post a message on the mailing list. Very little is known about such individuals and it is difficult, if not impossible, to contact them. Such users of the open source software cannot be contacted directly since the open source community does not keep a record of who downloads the software; such users of the gated software have signed a confidential licensing agreement with the corporate sponsor.

[5] Anyone can download the open source community's source code onto their own computer and make alterations to their private copy of the code, but making changes to the publicly available copy of the code, called the source code, is restricted to select individuals, called committers. Any participant can nominate an individual for committer status. If at least three existing committers voice support for the candidate and no existing committers voice a lack of support, the individual receives committer privileges.

activities. In describing their activities, subjects tend to volunteer information related to motives. At the conclusion of the interviews, interviewees were asked to comment on motives for participation that they had not brought up.

Interviews were generally conducted by telephone and recorded to facilitate data analysis. Interviews ranged in length from 30 to more than 120 minutes (90 minutes was the average). Interviewees were primarily college-educated males in their midtwenties to forties. The majority held full-time jobs as software developers or engineers. A few were employed by software-consulting firms and a handful were independent contractors.

Data Analysis

From the interviews I constructed categories based on the principles of grounded theory building (King et al. 1994, Strauss 1987). The construction of categories is an iterative process intended to create a common meaning that captures the essence of multiple observations (Locke 2001). After a category was named, I examined the data again and looked for other fragments of data (such as interview quotes) that fell within the category in a positive or negative way. If no other instances (positive or negative) appeared, the category was abandoned or revised. Alternatively, frequently mentioned categories were refined by adding specific descriptors. After identifying and refining a number of categories, I tried to understand how the different categories fit together into a coherent picture. In addition, I made a series of comparisons at two levels: between individuals in the same community and across the two communities.

4. Findings

This study finds that there are two types of participants, each with different sets of motives, in the open source community: need-driven participants and hobbyist participants. In contrast, the gated source community is populated almost exclusively by need-driven participants. Table 2 provides a summary of key findings related to developer participation. The table segments developers according to their reasons for creating and contributing code. For each subset of participants, the level of individual participation, the relative number of participants of this type in the community, and the general knowledge of code content and structure is reported. Detailed information on each of these participant subsets and the interaction between them is reported in the remainder of this section.

Need-Driven Participation

Need Drives Choice of Software Program. When describing their initial choice to use the software, virtually all interviewees spoke of needing to use the software for work-related purposes. Many open source participants consciously made the decision to use open source software, rather than commercially available software, so they could view and change the code to best fit their own needs. In contrast, several gated source participants expressed reluctance to use the gated software. The terms of the license made it difficult for them to get permission from their managers and/or corporate legal departments to use the software. Most undertook comprehensive searches for other options before choosing to use the gated software. Many explained that the technical capabilities of the gated software were unparalleled and required to solve the problems that they were working on.

> [Name of gated source project] offers great functionality, but it was difficult to get my company's lawyers to approve the license. I had to beg and plead and convince them that this was the best way to solve the issues we were facing . . . even that did not work the first time. We had to go back and work to convince management and legal that this was the only way to effectively do what we needed to do. (Long-term participant, gated source community, United States)

Table 2 Framework for Understanding Developer Participation

Reason to create	Reason to contribute	Level of individual participation	Relative number of participants[a]	Knowledge of code content and structure
Need	Reciprocity; norms	Low	High	Generally limited to area of initial problem
	Future product improvements	Varies, depends on need	Moderate	Primarily in area of initial problem, may expand
	Desire to integrate own code into source code	Moderate to high	Low	Varies
	Career concerns	Low and often peripheral	Very low	Varies
	None (do not contribute)	Very low	NA[b]	Generally limited to area of initial problem
Fun, enjoyment[c]	Feedback	High	Low	Begins in area of initial problem, expands

[a]Based on mailing list and interview data.

[b]It is difficult to estimate the size of this set, because those who do not contribute may not participate in mailing list discussions and are thus not represented in the interview sample. Modifying the code without subsequently contributing those modifications to the community is acceptable under many open source licenses (e.g., the Apache Software License, the Berkeley Software Distribution License).

[c]Evidence of this motive found primarily in the open source community.

Table 3 Motives for Initial Participation

Reason for initial involvement	Open source	Gated source
Need	25	17
Other	1	2

Note. $n = 45$.

Need Drives Code Creation. The need for software-related changes, alterations, or assistance drives initial and ongoing participation (Table 3). Both the technical content of these needs and the extent to which participants must manipulate the software to meet their needs varies widely, ranging from small adaptations to simple bug fixes to creating a new feature. Participants initially searched for existing solutions to their problems and only when no solution was found did they create their own. Because a need exists, participants generally do not wait for others to solve the problem.

> I was using the software for work. It's excellent, but there was a feature that I wanted that was not there.... I searched the documentation and mailing lists for information and to see if I had overlooked something, finally I asked a question. That spurred some conversation and someone suggested a beautiful way to implement the idea. (Short-term participant, open source community, United States)

> I first poked around to see if someone else had done the work. I only code if I can't find an existing solution.... I search the email archives, sometimes the uncommitted code and bug reports too...to find out if someone else had the same problem and whether they solved it or found a work-around.... If I don't find anything or if I need clarification, I ask and take care of it myself. (Long-term participant, open source community, United States)

Need-driven participants are a highly heterogeneous group with respect to both needs and skills (Franke and von Hippel 2003 also find evidence of this pattern). As a group, they contribute many new ideas to the community.

The practice of making potential participants find and solve their own problems—rather than assigning tasks—effectively creates a screening mechanism favoring the entry of those with specific needs. For individuals with a variety of other motives, such as learning or establishing a reputation, finding a niche of interest in which they have the skills to be useful can be a time-consuming and frustrating exercise.

Motives for Contributing Code. As a motivating factor, need helps us understand why individuals create code but does not explain why many individuals contribute what they have created to the community.[6] The costs of contributing code are positive and often relatively high in terms of the time and effort, e.g., the code must be cleaned up and refined, thought must be put into what is useful to others and what might be particular to one's own needs, and comments and explanations must be composed. The primary reasons cited by need-based participants for contributing code include reciprocity, future improvements, source code commits, and career concerns (Table 2, Column 2).

Reciprocity. Many need-driven participants report that reciprocity, obligation, or a desire to conform to the norms of the community drove their contributions. They report, "Others helped me, so I should help them" and "This is what is done in [name of project]." The amount of overall activity generated through reciprocity is high because of the large number of participants motivated by this factor.

Contributions triggered by reciprocity generally occurred over a short time span and consisted of answering questions and contributing patches or small bits of code. Participants motivated by reciprocity rarely spent time refining their code before contributing it to the community. These participants generally did not know—and were not interested in knowing—whether their code contribution had been accepted and incorporated into the source code. They deployed the version of the code that they had initially downloaded and altered. After contributing, most removed themselves from the mailing lists and did not keep up with software-related developments.

> The community helped me a lot. I really appreciated their suggestions and help, so I tried to help others with questions they had for a while. After a few weeks, I stopped scanning the mailing list—it's a lot of time to keep checking and see who you can help! A few days later, I unsubscribed from the mailing list. (Short-term participant, open source community, United States)

Future Improvements: Assistance When Building a Better Mousetrap. A number of need-driven participants reported a desire to find better solutions than the ones they created. By contributing their own work and ideas, they sought to (1) get feedback from others and, ideally, elicit subsequent improvements, (2) start or sustain discussions or development work that might be helpful to themselves and others, and (3) communicate that their need might be worthy of the attention of others. The knowledge of most participants in this

[6] Several individuals interviewed reported that they did not contribute all or part of their work product. It is difficult to estimate the relative size of this group based on the data available. The reasons given include: the work product was so specialized that it would be of little use to others, the participant lacked time, and the code was of competitive importance to their firm and hence would be kept proprietary.

group is limited to the area(s) of the code where they experienced a problem.

In addition, these participants regularly monitor mailing list postings for information related to their own needs. As they scan, they observe the questions of others and—due to their relatively deep understanding of one or more software modules—they find that they can provide others with assistance with very little effort. They report providing assistance due to reciprocity and/or a desire to cultivate more developers who might be able to assist them in the future.

Some of these participants become committers, particularly those who contribute high-quality code and/or undertake extended work that other committers deem widely useful. As committers these participants focus on work that directly furthers their own interests or takes a relatively small amount of time and effort to complete. Several report that, when first made a committer, they felt an obligation to "do more for the project." They undertook a few additional tasks for a short time frame, and then went back to focusing on their own needs.

Desire to Have Code Incorporated into the Source Code. Observers of the open source software phenomenon have suggested that participants seek to incorporate code into the source code so that their functional needs will continue to be met as the software evolves (Raymond 1999). Few participants engaged in this strategy. Instead, participants explained that what was important was *having* a particular feature or function. Many report that they avoid upgrading to new releases, relying on their individualized version for as long as possible.

The few participants engaged in this strategy tended to be employees of companies with strategic interests in the software or independent consultants. This set of developers expected to use the source code repeatedly for different projects, making it worthwhile for them to expend the time and effort required to incorporate changes into the source code. Some of these participants become committers. As committers, they make an effort to maintain the code in areas of strategic interest. They take the needs of other participants into account to avoid conflict, but their focus tends to be on their own requirements.

> We generally work on things and make changes that are in our best interest. We try and consider the interests of others, but really, most things we do in the projects we're involved with aren't controversial.... Otherwise we may have to deal with a lot of unhappy people and one of them will change the source code back anyway...if it's the best thing to do, you convince. If you can't convince, it may only be best for you...you can change your own version. (Long-term participant, open source project, United States)

Career Concerns. Reputation, skill development, and other career concerns are of relatively low importance in understanding the creation and contribution of code. In fact, these motivations were rarely mentioned. This led me to inquire about these motives at the end of each interview. Participant responses convinced me that their narratives were authentic and represented their reality. For example, when I brought up skill development and learning, many interviewees pointed out that their work consisted primarily of solving a small problem or two—hardly enough to build or hone a particular skill. Others corrected me and explained that what they were gaining was project-specific knowledge, not new skills.[7] When discussing the possibility of career benefits, most developers explained that they did not envision a link between participation and a better job and that, even if a link existed, their contributions were relatively small. The vast majority of interviewees reported *not* listing their open source work on their resume.

Career concerns can however generate positive outcomes for individuals and the community. For example, in two cases, developers who initially joined due to a need for the software found themselves unemployed and reluctant to reenter the corporate world. They subsequently chose to write software manuals and documentation for the project on behalf of for-profit publishers. While this documentation work did not directly contribute to the technical development of the code, it benefited the community by attracting and supporting new users and provided the developer with pecuniary benefits.

Hobbyist Participation in the Open Source Community

Fun and Enjoyment as Drivers of Code Creation. Over half of long-term open source participants describe their open source work as a fun and challenging hobbylike activity (Table 4, Column 2).

> I don't watch TV or sleep enough...this is my hobby.... I won't work a job that requires more than 40 hours.... I want to have breakfast and dinner with my kids.... I work on open source after they go to bed. (Long-term participant, open source community, Australia)

[7] The distinction between the skills to write code and knowledge of a particular piece of code is critical. This distinction corresponds to the difference between the ability to read, and knowledge of, a particular section of the *Odyssey*. Most participants report that they joined the project with the required technical skills and that they learned about the structure of the open source code as they worked. Some reported sharpening or expanding their skills, primarily through interactions with talented developers, practice, and viewing high-quality code.

Table 4 Motives for Long-Term Participation

Motive	Open source	Gated source
Need	7	14
Fun and enjoyment	11	1
Other	1	1

Note. $n = 35$.

The website of the open source community even includes a list of haikus written by developers. The following example evokes the implicit tension between a seductive hobby and legitimate work.

> Time, too much have you
> Major geeks these people are
> Boss know you do this? :)

The tasks undertaken by hobbyists vary widely, however all emphasize that they choose their own tasks and set their own schedules.

> I pick and choose the work that's most interesting to me . . . it's great when you find a challenging problem to work on—either on your own or because someone needs it—you can spend hours on it . . . . The routine stuff is okay, but I don't do much unless I just want to hack for a while and there are no really interesting problems around . . . . When I get bored, I will leave . . . . (Long-term participant, open source community, France)

Many described the freedom and creativity they experienced in defining and managing their open source work in great detail, contrasting this to the more structured and regimented work environment found in their day jobs.

"Interesting work" was self-identified or selected based on the needs of others. Hobbyists described many instances where they identified interesting challenges in the course of scanning mailing lists, bug reports, and contributed code. As they scan, they observe ongoing issues and view requests and suggestions. Sometimes a topic catches their interest and leads the participant to undertake a new task or even delve into a new area of the code. Hobbyists explain that, over time, their knowledge of the content and structure of the code accumulates, allowing them to undertake more difficult challenges, including those that require an understanding of multiple areas (modules) of the code.

Attention to technical detail is important to these participants. They are interested in striking a balance between meeting the needs of users and keeping the code simple, elegant, modular, and backward compatible.[8] Adherence to these principles (ideally) precludes the need for voluminous documentation, limits the number of standard features, and allows other participants to independently understand and alter the code.[9]

> Sometimes you work on an area and you notice that the code is getting more and more complicated . . . because many small changes have been made. There comes a time when you must start from scratch and rewrite the code with all the new functionality in mind. (Long-term participant, open source community, Germany)

> Could we use more documentation? Well, yes . . . but not in the way you are thinking . . . . In a perfect world [chuckle] we end up with clean code with annotations—short notes that developers understand. No documentation book required! (Long-term participant, open source community, United States)

Conversations with hobbyists and need-driven participants all indicate that hobbyists are the ones who tend the code and undertake most of the "maintenance work"—such as committing code, rewriting sections of the code, designing new releases, and fixing bugs—required to keep the project functioning. Because hobbyists choose their own tasks and set their own schedules, tasks sometimes go undone. For example, backlogs of code embodying new features or improving existing features often build up. The backlogs are generally not dealt with until one or more individuals signal that it is time to trigger a new code release, and takes primary responsibility for making sure that the release happens. Deadlines are virtually unheard of in the community unless self-generated by the participant who has taken on the work.

One might wonder how software development can qualify as a fun and engaging activity. Interviewees explained that when writing code, they were fully engrossed in solving a challenging puzzle, and offered analogies to playing chess, rock climbing, and putting together a difficult jigsaw puzzle. In each case, they expressed that part of the satisfaction in programming lay in the knowledge that a solution exists and that the solution could be found and implemented with creativity and patience. Many hobbyists consider structuring and writing code to be an artistic pursuit.

[8] Baldwin and Clark (2000) discuss the importance of modularity for product development more generally.

[9] In contrast, individuals motivated only by need might be tempted to create a feature and tack it on to the code. Successive additions of this kind are likely to result in code that is large, difficult to understand and improve, and suffers from problems due to interactions between areas of code. The process of writing an academic paper is analogous. If one writes a paper and asks several friends to comment on it and then merely "adds in" each individual's comments, the paper will likely be a mess. Instead, the comments must be understood, selected, and carefully integrated into the paper—and portions of the paper may have to be completely rewritten.

Several hobbyists engage in other creative pursuits such as music, creative writing, and art. These hobbyists are not alone, a number of accounts describing the careers and interests of engineers and technical professionals report that a subset of these individuals truly enjoy and find beauty in what they do and the products they create, and actively seek out opportunities to engage in challenging and useful activities in a variety of domains (Bailyn and Lynch 1983, Gelernter 1998, Levy 2001, Moody 2001).

Hobbyists tend to be highly skilled and experienced software developers. Many hold managerial positions in the companies where they work. Many reported that their work activity is not sufficiently interesting or engaging, and that open source software development provides a venue in which they can satisfy their desire to be creative and solve challenging puzzles.

Hobbyists are not participating to develop skills or build their resumes. When I inquired about these motives at the end of each interview, I received a variety of spontaneous and often entertaining responses. For example, when I brought up skill development, hobbyists quickly corrected me and explained that they already possessed a stockpile of skills. Several hobbyists explained that they actively tried to manage perceptions of their open source work with their managers, because they did not want to be accused of neglecting their day jobs. When asked whether they listed their open source work on their resumes, many hobbyists laughed and informed me that they did not, while assuring me that in the context of their work experience, their open source activities were trivial. Only one hobbyist listed his open source work on his resume—he was a student and explained that this was his primary hobby (interestingly, he also explained that in the country where he lived, the programming language used in the community was rarely used in corporate software development, and thus unlikely to help him land a job).

Hobbyist Motives for Contributing Code: Feedback. Enjoyment derived through the act of programming does not necessitate that one work within a community—one could work alone. Working within a community helps the hobbyists identify tasks that they find challenging and interesting, and that are useful for others. Contribution is necessary to obtain feedback affirming that one's activities are useful to others.[10]

> Why work on something that no one will use? There's no satisfaction there. (Long-term open source participant, United States)

[10] This pattern is in line with psychological theory that finds that informational feedback enhances intrinsic motivation (Amabile 1983; Boggiano and Pittman 1992; Deci 1971, 1975; Kruglanski 1975; Lepper and Greene 1978).

One hobbyist even reported that he and two other developers had recently taken a day off from their jobs to present the open source project to a firm that was considering using the project's software. When asked why, he explained:

> It is rewarding when you see that what you helped create is used by many people. I want to let many people know about this software and I want them to use it. (Long-term open source participant, Germany)

Hobbyists report monitoring the mailing lists, bug reports, and contributed code for discussions related to their contributions. They tend to be highly receptive to bug reports and well-formulated questions and suggestions. Such feedback can create interesting puzzles or challenges, improve the code, and affirm the usefulness of what they have created.

Many hobbyist participants are also committers. Attaining "committer" status may be an additional form of feedback in the open source community, letting the participant know that their contributions are valued and signaling what existing committers value to others, thereby reinforcing norms. Although it is possible that such recognition affects contribution decisions, most committers viewed the designation as a welcome "pat on the back" that made it easier for them to alter the code, rather than something they diligently worked to attain (note that many committers leave the project after several months, see final subsection of this section).

Contextual Factors Influencing Hobbyist Participation: Control and Fit. Hobbyist participants report that the level of control exercised in the community and the behaviors of other participants influence their choice to participate. Heavy-handed control can deter participation:

> Sometimes [the participation decision] isn't [all] about technical considerations. There's another OS project whose technology I use and I want to develop further, but the "benevolent dictator" is simply a dictator . . . the few developers who stick around are like that too . . . who needs that? This project is flexible . . . . I'm watching for interesting stuff to do here. (Short-term participant, open source community, United States)

Not surprisingly, hobbyists will choose not to work in project communities where they feel uncomfortable.

> [Name of open source project] is really nice—not only professional, but nice . . . look at [name of another project sponsored by the same community], they are a lot rougher . . . . Those developers are extremely talented, more talented than the ones here . . . but, it's just not my thing, to listen to people carry on like that. (Long-term open source participant, United States)

A Symbiotic Relationship: Need-Driven Participants and Hobbyists

Need-driven participants and hobbyists within the open source community differ in their motivations and actions, yet each group benefits from the other. The questions, suggestions, and contributions of need-driven participants feed hobbyists searching for interesting and useful challenges. Hobbyists, in turn, provide assistance and support to need-driven participants by answering questions, creating desired features, integrating need-driven contributions into the source code, and providing other "maintenance" services that, for example, preserve or improve the architecture, modularity, and backwards compatibility of the code. This interaction appears to be critical in allowing the open source project to function without a formal task identification and assignment system.

This does not, however, mean that every request is satisfied. In fact, several cases were observed where requests for information or assistance on the community mailing lists received no response and several ideas deemed "worthy" by a number of participants interacting on the mailing list were not implemented due to either lack of interest or time.

Volunteers Behave Differently in Open Source and Gated Source Communities

Although the open source community greatly benefited from the presence of hobbyist developers, virtually all long-term gated source participants continued to focus on need as their primary reason for project involvement (Table 4, Column 2).[11]

In describing their activities within the gated source community, participants voiced blatant disapproval of two elements of the governance structure: The level of code control held by the sponsor (the sponsor is the only actor able to make changes to the source code) and ownership by the corporate sponsor.[12] Each of these is described in greater detail below.

Day-to-day control over the code made participants question whether their efforts would go to waste, as their needs and activities might not be reflective of the sponsor's financial interests:

> I answer questions and stuff, but I don't feel the need to contribute my changes to the community. It's time consuming and I don't know if [the corporate sponsor] will do anything with it . . . . At the end of the day, they make the decisions with their commercial licensees in mind. (Long-term participant, gated source community, United States)

Ownership by the corporate sponsor raised four issues. The first issue involves unrestricted use of the code by the participant:

> Why should I contribute to something that is not mine? . . . It's okay if it's mine and someone else's and someone else's, but I have to be able to use it the way I want, whenever I want. (Short-term participant, gated source community, United States)

The second issue involves the fairness of restrictions on wider code use and distribution, as illustrated below.

> I make the changes that I really need and so does everyone else and we benefit from one another . . . . There are a lot of things the project still needs that I keep asking [the corporate sponsor] to develop . . . they are not absolutely critical, but they'd take the software to the next level and expand its capabilities . . . if I develop it and then [the corporate sponsor] says I can't let others see it or work on it or use it in whatever way that makes sense—now come on! That's not how it works! (Long-term participant, gated source community, United States)

The third issue involves various scenarios under which the sponsor might inadvertently or deliberately, in the words of a participant, "appropriate code away" from the community. For example, conflicts over ownership and use might arise if the sponsor goes out of business, chooses to discontinue the project, or wants to sell the code. The fourth issue involves problems created by the commercial-use clause of the license. Although the code may be used freely for developmental work by other firms, commercial use requires the negotiation and payment of a royalty fee to the sponsor. Many participants were wary that the sponsor might stipulate outrageous license terms for commercial use after the participant (on behalf of an employer or himself) invested in the creation of a product based on the code.

Due to these issues, those who participated tended to be developers with no other option but to use

[11] Conversations with several gated source participants who attended or presented at the conference underscored the fact that participation was motivated more by individual benefit and less by fun and enjoyment. For example, when asked why they took the time to present their work at a conference, one participant explained that in order to get an office with a window, she had to get a promotion. The remaining step toward earning a promotion was to make a technical presentation at a conference. Another explained that he was running a consulting business based on the technology and hoped to make contacts with potential clients during the conference.

[12] Alternative explanations might be that the gated source software is inherently less challenging or that volunteers are relying on employees of the corporate sponsor to do work, however, (1) the gated source project is generally regarded by software developers to be more exciting and revolutionary than the open source project. It does not appear that there is any lack of fun or challenge to be had. (2) While it is true that gated source community participants might desire that the corporate sponsor do as much work as possible, especially the more mundane work, this does not explain why volunteers are not seeking out the work that they view as fun and enjoyable.

the gated source software, those who believed that they could trust the sponsor (e.g., the sponsor's existing customers), and those willing to work under the licensing constraints (e.g., start-ups and consultants who believed they would gain financially from the technology). Many of these participants chose not to contribute the code that they created, leaving other developers with less to work with and build on. In the long run, this limits cumulative development activity and overall value creation. When these participants did contribute, their contributions often stemmed from strategic concerns.

Leaving the Community

Most open and gated source developers—even hobbyists and those who attain committer status—appear to leave the project within one year. Participants are not expected to remain on the project indefinitely and exit is understood to be a normal part of the process, even for committers and hobbyists.[13]

> Most [committers] stick around for maybe 3–4 months at most, it's okay to leave.... The ones that stay for over 6 months tend to really stay...for a year or two. (Long-term participant, open source community, Australia)

5. Discussion

Summary of Key Findings

This paper contributes three key findings to the literature. First, open source software participants join the community to satisfy a need, but many of those who continue to create code do so because they enjoy programming. These hobbyist developers are critical to the functioning of the open source community. Second, reciprocity is an important factor driving the contribution of code to the community. Third, the governance structure of the community dramatically affects the participation choices of volunteer software developers. Below, I discuss limitations of the study. This is followed by a discussion of each of the key findings.

Limitations

There are limitations to be considered when interpreting and using the results of this study. The use of in-depth, qualitative data offers the opportunity to gain understanding and build theory in areas that we understand little about, however such theory runs the risk of being idiosyncratic and not generalizable to the entire population (Eisenhardt 1989). In this case, the software developed in both projects is generally used by software developers and engineers working within corporations. Software developers and users in other projects might have different individual characteristics and motivations for creating and contributing, both stemming from and resulting in different structural arrangements. For example, one might observe that groups of ideologically driven developers—in their effort to displace existing proprietary software products—will channel their efforts toward creating software that mimics the functionality of the existing software, rather than create software that serves an altogether novel purpose. As a second example, one might expect slight differences in the development processes for software projects that are used by both technologically savvy users and the general public. In such projects, hobbyists may need to be more selective about the challenges they choose and feedback might be constrained as many users lack the technical vocabulary required to communicate effectively with developers. In fact, in projects such as the Firefox Web browser, we see technological solutions, such as the automatic reporting of crashes or bugs back to the community, being deployed to help alleviate this problem. Finally, care should be taken when using findings from this paper to better understand participation in innovation communities operating in other product domains.

Hobbyists

Theories of innovation generally assume that either financial incentives or need-based incentives drive innovative activity. Here we see strong evidence for a third source of motivation: fun and enjoyment derived from the very act of creating and tinkering.

The individual's relationship to the task and the social structure in which the individual is embedded is likely to influence the motives that drive innovative activity. For example, professionals may respond to financial incentives, scientists may strive to increase their status, and hobbyists may be largely driven by enjoyment. Moreover, the same individual might act differently and be driven by different motives in different contexts, e.g., consider the activities and behaviors of the hobbyist developer at work versus in his spare time. Differences in motives might also create differences in the types of innovations created. For example, those seeking financial rewards might devote their efforts to designs likely to be of interest to a large market segment, thereby innovating along dimensions known to be important to consumers. In contrast, those driven by enjoyment and challenge

[13] A few individuals within the open source umbrella project were observed to actively participate in the community for more than two years. Some of these participants take an active role in the management of the umbrella community; further investigation of their motives and role in enabling the functioning of the community would further illuminate the structure of the community.

might seek to explore uncharted territory or very specific areas of inquiry, thereby creating innovations that are functionally novel.[14]

An individual's motivation toward a task may shift, as was observed among the open source hobbyist participants. This is one of the few empirical settings where intrinsic motivation is built and reinforced—"crowded in," by the social structure (Osterloh and Frey 2000).[15] Future research is needed to examine the process by which the shift in individual-level motives occurs, investigating in particular whether a selection mechanism that favors those with hobbyist tendencies exists or whether interaction with the community leads to a shift in the individual's identity and self-perception.[16]

Property Rights, Fairness, and Reciprocity

This study highlights the importance of fairness in supporting exchange relationships. In this setting, property and decision-making rights affected individuals' perceptions of fairness, which in turn affected their behaviors. Participants in each community were aware of who controlled property and decision-making rights and appear to view that actor as their primary exchange partner. That is to say, when assessing reciprocity within the community and choosing the extent to which they wish to create and contribute code, open source participants focus on the actions of volunteer community members, whereas gated source participants focus on the actions of the corporate sponsor.

The possibility of opportunistic ("unfair") actions by those holding control rights can both decrease and alter the character of volunteer participation. Gated source developers feared that the corporate sponsor might exercise control rights in a strategic manner benefiting the corporation over the community. As a result, gated source participants invested effort only if they very much needed the software and were less likely to contribute their findings to the community. Moreover, they were unwilling to engage in fun and challenging development activities, depriving themselves of an enjoyable activity and depriving the community of their efforts and talents. From the perspective of behavioral game theory and evolutionary psychology, it is not surprising that perceptions of fairness weigh heavily into an individual's decision to work with others and even leads individuals to punish others at a cost to themselves (Barkow et al. 1992; Kahneman et al. 1986a, b). From this perspective, the creation of a neutral and accessible commons is crucial for fostering community-based innovation. Keeping a resource in the commons both allows others to draw on the resource and mitigates the number of strategic games played by others (Lessig 2001, p. 72).

Implications for Hybrid Strategies

Firms are anxious to leverage the open source development model. For firms, community development offers the possibility to gain developmental assistance in noncritical areas and increase adoption (West 2003). Value appropriation requires the firm to define and control property rights. Unfortunately, activities that permit value appropriation by the firm are sometimes detrimental to value creation within the community. Here I discuss the impact of two broad sets of governance mechanisms—decision rights and property rights—on participation (Table 5 summarizes this discussion and distinguishes between observed and hypothesized patterns).

Decision-Making Rights.

Code Control. In the gated source community, only the corporate sponsor is allowed to alter the source code. This strict control over the code affects both need-driven and hobbyist participants. Need-driven participants worry that their voices will be drowned out by the needs of the firm and its customers when software-related decisions are made. Such control limits the ability of hobbyists to work and contribute in self-defined ways. In addition, the volume of feedback and overall activity is likely to decline due to both decreased participation and tighter control over what is committed to the source code and, therefore, used by others.

[14] Those driven by their own needs might pursue either well-known or novel dimensions, potentially creating innovations that are useful to many or only to themselves, and that extend existing functionality or create altogether new functionality.

[15] Cognitive evaluation theorists argue that intrinsic motivation is based on feelings of competence and self-determination—feelings supported by feedback in the form of information and rewards from external sources (Boggiano and Pittman 1992; Deci 1971, 1975; Deci et al. 1999; Lepper and Greene 1978). *Informational content* heightens feelings of competence and strengthens the feeling of internal control, raising ("crowding in") intrinsic motivation. In contrast, *controlling content* heightens feelings of being stressed from the outside and strengthens perceived external control, decreasing ("crowding out") intrinsic motivation. Additional research is required to understand the mechanisms by which the community encourages its members to provide informational content and quell controlling content.

[16] Collecting such data on the career progression of volunteer software developers will improve our understanding of the individual and institutional-level factors supporting community-based innovation (for examples and a discussion of research on careers, see Barley 1989, Becker 1963, Goffman 1959, Hogg and McGarty 1990). This information will aid in the design and management of innovation communities, and potentially in the design and management of other types of nonprofit organizations relying on volunteer labor. It may also improve our ability to design more satisfying work environments for engineers and other technical professionals.

Table 5 Governance and Development

Governance mechanism	Potential impact on collective development process
Decision-making rights	
Code control	• Decreases perception that needs of various actors will be met, thereby decreasing contribution[a] • Limits ability of hobbyists to work and contribute in self-defined ways[d] • Interferes with feedback processes[d]
Domination or control over mailing list interaction	• Inhibits voicing of heterogeneous requirements or viewpoints[b] • Severely inhibits feedback processes that hobbyists participants enjoy[b]
Property rights	
Private ownership of source code	• Creates the possibility that the developer will not be able to use the code at a later date[a, c] • Decreases perception that needs of various actors will be met, thereby decreasing overall activity and contribution[a] • Participation may be restricted to those with desperate need for code or who are willing to trust the firm[a] • Inhibits reciprocity[a]
Restrictions on use, modification, and distribution	
Broad restrictions on use and distribution	• Because developers feel entitled to use and share code they helped develop, this restriction deters developers and thereby decreases the size of the community and the likelihood of future product improvements and feedback[a]
Restrictions on commercial use and distribution; negotiated terms	• Potential for hold-up created as work will be done first and terms negotiated after a commercial product developed. Only those who trust the firm or are able to negotiate in advance are likely to participate[a]
Restrictions on commercial use and distribution; standard terms	• If terms are reasonable and fair, those with the ability to pay are likely to participate[d]
Proprietary modifications	
Allowed	• System appears to function fully[a, c]
Not allowed ("Free" software)	• System appears to function fully[c]

[a]Directly observed in this study.
[b]Indirectly observed in this study.
[c]Data based on other studies or observations; see text.
[d]Hypothesized relationship.

Domination of Mailing List Interaction. Firms may inadvertently dominate mailing lists through a desire to influence the direction of the project or because firm employees represent a substantial portion of the participant pool, as might be the case when a firm-sponsored community is newly created. This may act to inhibit developers from voicing heterogeneous views, resulting in decreased volunteer participation. This relationship was not directly observed in this study, however developers' distaste of tightly controlled open source projects was observed.

Property Rights.

Private Ownership of Source Code. Private ownership of the code acts to dismantle the collective development process in a variety of ways. Most noticeably, ownership by the firm creates the possibility that the developer will not have access to the code at a later date. Participants value the results of their efforts and expect to continue using the software well into the future. The open source project gave them this right, but the gated source project did not make this guarantee. Private ownership also appears to inhibit reciprocity: if the firm is not donating the code to the community, why should the developer take additional time and effort to donate code to the firm?

Restrictions on Use, Modification, and Distribution. The restrictions placed on the gated code with respect to commercial use, modification, and distribution all act to reduce the degrees of freedom that an individual can exercise when using and creating the code, particularly when compared to open source licensing arrangements. These restrictions can be thought of as limiting the value available to the individual developer, i.e., the developer can only use the code for certain purposes, modifications made and deployed must meet community standards (rather than his own preferences), and the code may only be shared with others willing to abide by the community's governance arrangements, thereby decreasing the volume of subsequent improvements and feedback that many developers relish. On the other hand, these restrictions might create value for the company.

Restrictions on commercial use in the gated source community created additional problems because licensing terms were negotiated with the firm on an individual basis. This decreased trust dramatically by creating the possibility of ex post hold-up problems. Restrictions on commercial use with terms set in advance and applicable to everyone may be less problematic.

Proprietary Modifications. A large part of the long-standing debate between free and open source software advocates concerns proprietary modifications. Software derived from free software cannot be made proprietary. The "copyleft" provisions of the GNU General Public License (GPL) and similar

licenses mandate that all derivative works be distributed under the same licensing terms as the original software, thereby ensuring that code remains free (Stallman 2001).[17] In contrast, the open source definition, while inclusive of free software licenses, allows proprietary modifications to be made (Open Source Initiative 2004). There exist examples of successful projects governed by both types of licenses, although GPL-style licenses appear to dominate the landscape (Lerner and Tirole 2005).

Developers using the GPL often focus on the importance of copyleft in the early years of community formation, when the developer was the only one—or one of a small group—working on the code (Stallman 1999):

> Someone could make an improved proprietary version and it could displace the free version. And as a result, people might be using my code but they would not have the freedom that I hoped they would have and I would not have it either, unless I kept using the inferior free version. But if nobody joined me, it would not do much good. And so . . . I worked out the idea of copyleft. (Richard Stallman as quoted in O'Mahony 2003, p. 1183)

It appears that the GPL was initially designed to institutionalize reciprocity in an attempt to seed a community. Licenses, however, are not required for reciprocity to operate as indicated by the fact that proprietary modifications were allowed within the open source community, yet many participants contributed based on a norm of reciprocity and because development is concentrated within the community. It is likely that this norm is established early in projects that do not use GPL-style licenses. Future research might investigate how such norms are initially established and under what circumstances copyleft licenses, in addition to norms, are necessary to ensure contribution.

Governance practices can create shortcomings in hybrid forms, although some shortcomings can be ameliorated. For example, the firm sponsoring the gated source community has significant resources at its disposal and employed a team of developers and marketers to work on the gated source project. These employees took care of many of the tasks—such as assisting participants, incorporating suggestions and code, and maintaining the overall architecture of the code—that would have been fulfilled by volunteer hobbyists in the open source community.

6. Conclusion

Innovation communities represent a novel model for innovation development that has been documented in a number of diverse product areas. Our knowledge of this novel structure and the societal benefits it creates is growing, but many questions and issues remain to be explored. In this paper, I investigate issues pertaining to participation and governance in two software communities. I find that fun and challenge are key drivers of participation for hobbyists, a group that is critical to the functioning of the open source community. In addition, reciprocity and fairness are important issues, strongly affecting the willingness of volunteer developers to contribute to the community. Firms seeking to construct hybrid arrangements that balance community-based value creation with private value appropriation may encounter difficulties: the very mechanisms that allow them to appropriate private benefits may deter participation.

[17] GNU is based on the recursive acronym "GNU is not Unix" (Stallman 1999).

Acknowledgments

I wish to thank my father, Kalpesh R. Shah, for his support, advice, and love. Special thanks to Rajshree Agarwal, Carliss Baldwin, Roberto Fernandez, Geoff Love, Eric von Hippel, Sidney Winter, the two anonymous referees who provided thoughtful comments on earlier versions of this paper, and the many software developers who graciously shared their time and experiences.

References

Allen, R. C. 1983. Collective invention. *J. Econom. Behav. Organ.* **4** 1–24.

Amabile, T. M. 1983. *The Social Psychology of Creativity*. Springer-Verlag, New York.

Bailyn, L., J. Lynch. 1983. Engineering as a life-long career: Its meaning, its satisfactions, its difficulties. *J. Occupational Behav.* **4** 263–283.

Baldwin, C., K. Clark. 2000. *Design Rules*. Harvard Business School Press, Cambridge, MA.

Barkow, J. H., L. Cosmides, J. Tooby. 1992. *The Adapted Mind: Evolutionary Psychology and the Generation of Culture*. Oxford University Press, New York.

Barley, S. R. 1989. Careers, identities, and institutions: The legacy of the Chicago School of Sociology. M. B. Arthur, D. T. Hall, B. S. Lawrence, eds. *Handbook of Career Theory*. Cambridge University Press, New York, 41–65.

Becker, H. S. 1963. *Outsiders: Studies in the Sociology of Deviance*. Free Press, New York.

Becker, H. S. 1998. *Tricks of the Trade: How to Think About Your Research While You're Doing It*. University of Chicago Press, Chicago, IL.

Boggiano, A. K., T. S. Pittman. 1992. *Achievement and Motivation: A Social-Developmental Perspective*. Cambridge University Press, New York.

Coleman, J. S. 1994. A vision for sociology. *Soc. Sci. Modern Soc.* **32**(1) 29–34.

Deci, E. L. 1971. Effects of externally mediated rewards on intrinsic motivation. *J. Personality Soc. Psych.* **18** 105–115.

Deci, E. L. 1975. *Intrinsic Motivation*. Plenum, New York.

Deci, E. L., R. Koestner, R. M. Ryan. 1999. A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. *Psych. Bull.* **125** 627–668.

Denzin, N., Y. Lincoln. 2000. *Handbook of Qualitative Research*. Sage, Thousand Oaks, CA.

Eisenhardt, K. M. 1989. Building theories from case study research. *Acad. Management Rev.* **14**(4) 532–550.

Ferris, T. 2002. *Seeing in the Dark: How Backyard Stargazers Are Probing Deep Space and Guarding Earth from Interplanetary Peril*. Simon & Schuster, New York.

Franke, N., S. Shah. 2003. How communities support innovative activities: An exploration of assistance and sharing among end-users. *Res. Policy* **32** 157–178.

Franke, N., E. von Hippel. 2003. Satisfying heterogeneous user needs via innovation toolkits: The case of apache security software. *Res. Policy* **32** 1199–1215.

Franz, K. 2005. *Tinkering: Consumers Reinvent the Early Automobile.* University of Pennsylvania Press, Phildelphia, PA.

Freiberger, P., M. Swaine. 2000. *Fire in the Valley*. McGraw-Hill, New York.

Gelernter, D. 1998. *Machine Beauty*. Basic Books, New York.

Ghosh, R. A. 1998. First Monday interview with Linus Torvalds: What motivates free-software developers. *First Monday* **3**(3), http://www.firstmonday.org/issues/issue3_3/torvalds/.

Ghosh, R. A., R. Glott, B. Krieger, G. Robles. 2002. Free/libre and open source software: Survey and study. Report, International Institute of Infonomics, University of Maastricht, Maastricht, The Netherlands.

Giddens, A. 1984. *The Constitution of Society: Outline of the Theory of Structuration*. University of California Press, Berkeley.

Goffman, E. 1959. The moral career of the mental patient. *Psychiatry* **22**(2) 123–142.

Hann, I. H., J. Roberts, S. Slaughter, R. Fielding. 2002. Delayed returns to open source participation: An empirical analysis of the Apache HTTP Server Project. Working paper, Carnegie Mellon University, Pittsburgh, PA.

Healey, K., A. Schussman. 2003. The ecology of open source software development. Working paper, University of Arizona, Tucson, AZ.

Hertel, G., S. Niedner, S. Hermann. 2003. Motivation of software developers in open source projects: An Internet-based survey of contributors to the linux kernel. *Res. Policy* **32** 1159–1177.

Hogg, M. A., C. McGarty. 1990. Self-categorization and social identity. D. Abrams, M. A. Hogg, eds. *Social Identity Theory: Constructive and Critical Advances*. Harvester/Wheatsheaf, New York, 10–27.

Jencks, C., L. Perman, L. Rainwater. 1988. What is a good job? A new measure of labor-market success. *Amer. J. Sociol.* **93**(6) 1322–1357.

Jensen, M., W. Meckling. 1994. The nature of man. *J. Appl. Corporate Finance* **7**(2) 4–19.

Jeppesen, L. B., M. J. Molin. 2003. Consumers as co-developers: Learning and innovation outside the firm. *Tech. Anal. Strategic Management* **15**(3) 363–384.

Kahneman, D., J. Knetsch, R. Thaler. 1986a. Fairness and the assumptions of economics. *J. Bus.* **59**(4) S285–S300.

Kahneman, D., J. Knetsch, R. Thaler. 1986b. Fairness as a constraint on profit seeking: Entitlements in the market. *Amer. Econom. Rev.* **76**(4) 728–741.

King, G., R. Keohane, S. Verba. 1994. *Designing Social Inquiry*. Princeton University Press, Princeton, NJ.

Kline, R., T. Pinch. 1996. Users as agents of technological change: The social construction of the automobile in the rural United States. *Tech. Culture* **37** 763–795.

Kruglanski, A. W. 1975. The endogenous-exogenous partition in attribution theory. *Psych. Rev.* **83** 387–406.

Kuan, J. 2001. Open source software as consumer integration into production. Working paper, SSRN, http://ssrn.com/.

Lakhani, K., E. von Hippel. 2003. How open source software works: Free user to user assistance. *Res. Policy* **32**(6) 923–943.

Lepper, M. R., D. Greene. 1978. *The Hidden Costs of Reward*. Erlbaum, Hillsdale, NJ.

Lerner, J., J. Tirole. 2002. The simple economics of open source. *J. Indust. Econom.* **52**(June) 197–234.

Lerner, J., J. Tirole. 2005. The scope of open source licensing. *J. Law Econom. Organ.* **21** 20–56.

Lessig, L. 2001. *The Future of Ideas*. Random House, New York.

Levy, S. 2001. *Hackers: Heroes of the Computer Revolution*. Penguin Books, New York.

Locke, K. 2001. *Grounded Theory in Management Research*. Sage, Thousand Oaks, CA.

Moody, G. 2001. *Rebel Code: Inside Linux and the Open Source Revolution*. Perseus Publishing, Cambridge, MA.

O'Mahony, S. 2003. Guarding the commons: How community managed software projects protect their work. *Res. Policy* **32**(7) 1179–1198.

Open Source Initiative. The open source definition. http://www.opensource.org (accessed August 2004).

Osterloh, M., B. Frey. 2000. Motivation, knowledge transfer and organizational forms. *Organ. Sci.* **11**(5) 538–550.

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

Shah, S. 2000. Sources and patterns of innovation in a consumer products field: Innovations in sporting equipment. Working Paper 4105, Massachussetts Institute of Technology, Sloan School, Cambridge, MA.

Shah, S. 2005. Open beyond software. C. Dibona, D. Cooper, M. Stone, eds. *Open Sources 2: The Continuing Evolution.* O'Reilly Media, Sebastopol, CA.

SourceForge. http://sourceforge.net (accessed October 2005).

Spradley, J. 1979. *The Ethnographic Interview*. Holt, Rinehart & Winston, New York.

Stallman, R. 1999. The GNU operating system & the free software movement. C. DiBona, S. Ockman, M. Stone, eds. *Open Sources*. O'Reilly, Sebastopol, CA, 53–70.

Stallman, R. 2001. Philosophy of the GNU project. http://www.gnu.org/philosophy/ (accessed August 2001).

Strauss, A. 1987. *Qualitative Analysis for Social Scientists*. Cambridge University Press, New York.

Van Maanan, J. 1998. Different strokes: Qualitative research in the administrative science quarterly from 1956 to 1996. J. Van Maanan, ed. *Qualitative Studies of Organizations*. Sage Publications, Thousand Oaks, CA, ix–xxxii.

von Hippel, E., G. von Krogh. 2003. Open source software and the "private-collective" innovation model: Issues for organization science. *Organ. Sci.* **32**(2) 209–233.

Weizenbaum, J. 1976. *Computer Power and Human Reason: From Judgment to Calculation*. W. H. Freeman, San Francisco, CA.

West, J. 2003. How open is open enough? Melding proprietary and open source platform strategies. *Res. Policy* **32**(7) 1259–1285.

Wilson, J. 2000. Volunteering. *Annual Rev. Sociol.* **26** 215–240.