# CHAPTER 4

## UNDERSTANDING THE NATURE OF PARTICIPATION & COORDINATION IN OPEN AND GATED SOURCE SOFTWARE DEVELOPMENT COMMUNITIES

SONALI SHAH

## CONTENTS

4.1 Introduction....4
4.2 Literature Review....7
*4.2.1 Community-Based Innovation & Product Development: A Different Mode of Organizing*....7
*4.2.2 Motivations for Two Aspects of Community Participation: Creating and Contributing*....8
4.3 Method, Setting & Data....15
*4.3.1 Factors Guiding Choice of Setting*....15
*4.3.2 Data Collection & Analysis*....17
*4.3.3 Operational Definitions*....19
4.4 Findings....19
*4.4.1 Characteristics of Individuals in the Sample*....20
*4.4.2 Modes of Communication*....22
*4.4.3 Reasons for Creating & Analyzing Information Evolve Over Time*....22
*4.4.4 Reasons for Contributing Information*....30
*4.4.5 Achieving Community "Coordination"*....36
4.5 Discussion....44
*4.5.1 Fairness: Responses to Contribution & Governance*....44
*4.5.2 How Do Communities Turn "Ad Hoc" Contributions Into High Quality Products?*....46
4.6 Model....46
*4.6.1 Need Drives Initial Participation*....47
*4.6.2 Reasons for Participation Change Over Time*....48
*4.6.3 Enjoyment & Personal Satisfaction Drive Many Long-Term Developers*....50
4.7 Conclusion....54
Appendix....56
Bibliography....61

## TABLES

Table 4-1: Categories of Motivations for Participating In Innovation & Product Development Communities....10
Table 4-2: Reasons for Participation as Rated By Linux Participants....11
Table 4-3: Sample Characteristics....21
Table 4-4: Need for Software Drives Initial Volunteer Involvement....24

Table 4-5: Motives in Gated Communities Do Not Appear To Change Over Time ..... 25
Table 4-6: For Those Why Stay In Os Communities, Motives Evolve Over Time ..... 25
Table 4-7: Long-Term Participants Without Own Need For Software Describe Themselves as Motivated By Challenge & Fun (Includes Os & Gated Project Participants) ..... 27
Table 4-8: Some Collective Action Issues and Responses ..... 43
Table 4-9: Linkage between Creation, Contribution, and Knowledge of Code Structure ..... 52
Table 4-10: Participation Models In Firms vs. Communities ..... 53

**FIGURES & CHARTS**

Figure 4-1: Evolution of Primary Motives over Time ..... 49
Figure 4-2: Proprietary versus Community Software Development ..... 60
Figure 4-3: Two Approaches to Community Software Development ..... 60

# CHAPTER 4

## UNDERSTANDING THE NATURE OF PARTICIPATION & COORDINATION IN OPEN AND GATED SOURCE SOFTWARE DEVELOPMENT COMMUNITIES

SONALI SHAH

**Abstract**

Voluntary product development communities have no paid staff or management and may even be geographically dispersed, yet they provide participants with a social context and the resources to create useful products that have, on occasion, displaced or significantly improved-upon commercially-produced products. Such communities represent a very different type of organizational structure for innovation – a structure that we understand little about. This paper explores the motivations of participants from two software development communities and finds that most participants are motivated by either a need to use the software or an enjoyment of programming. The latter group, hobbyists or enthusiasts, are critical to the long-term viability and sustainability of open source software code: they take on tasks that might otherwise go undone, are largely "need-neutral" as they make decisions, and express a desire to maintain the simplicity, elegance, and modularity of the code. The motives of hobbyist evolve over time; most join the community because they have a need for the software and stay because they enjoy programming in the context of a particular community. Governance and licensing structures affect this evolution.

## Section 4.1: Introduction

Individuals are voluntarily contributing their time, skill, and energy to innovation and product development communities. These communities have no paid staff or management and may even be geographically dispersed, yet they provide participants with a social context and the resources to create useful products that have, on occasion, displaced or significantly improved upon commercially-produced products. Such communities represent a very different type of organizational structure for innovation – a structure that we understand little about.

This paper focuses on one aspect of community-based product development: Why do participants voluntarily work within and contribute to communities that provide, support, and maintain a public good? This paper takes an inductive approach and develops a model to understand participation in product-development communities. Participants of two large and well-known software development communities are studied. One community is "open source" and the other is "gated source" – this means that the communities are governed by different institutional structures and intellectual property licensing arrangements, although both communities seek to attract the efforts of volunteer software developers and allow source code (not just binary code) to be distributed. This distinction is exploited and used to develop the model. It is important to note that, as used here, the terms open and gated source refer to two different and distinct modes of development; one is not a subset of the other[1]. Data were collected via interviews and analysis of publicly available project information.

Empirically, we see that two types of participants emerge. In the first, more populous group, a need to use the software drives product creation and improvement, while notions of fairness or the desire to benefit from the potential and subsequent improvements of others leads people to contribute what they know to the community. In the second, much smaller, group, participants do work that is largely unrelated to their own needs. These participants derive enjoyment from engaging in creative and

[1] A more detailed explanation of open source and gated source software development is provided in the Appendix.

challenging programming tasks, working with others, and seeing the software improve. They primarily undertake tasks that interest them, thus the tasks they choose are not always those for which the project has the greatest need or for which the most users express an interest. Nevertheless, feedback from users and developers of the code is also a vital component of the system for these developers. Over time, members of the second group of participants acquire a greater understanding of larger and larger portions of the software code. Many of these individuals express a dedication to keep the design of the software simple and understandable – so that others can continue to improve upon and "play with" the code. This desire to write and maintain an elegant codebase is critical to the continued viability of the community and code. The formal and informal processes of the open source community support these types and quantities of participation. Overall, the formal and informal processes of the communities allow individual contributions to be transformed into high-quality products without the need for paid or full-time staff or hierarchical mechanisms for task coordination.

Theory from the literature on fairness and social exchange in the fields of evolutionary psychology and behavioral economics, as well as theory and empirical findings from the innovation management literature, are used to understand and explain OS software development communities. The model developed in this paper helps us understand how high-quality products are generated without direct control over participant actions by a volunteer "staff"; how these individuals accumulate the knowledge required to manipulate the code[2]; and why talented software developers, most of whom hold full-time jobs and have significant demands on their time, are participating in open source development projects. The model also shows why different institutional structures may result in different amounts and types of participation, and is therefore useful to firms interested in creating communities to attract volunteer developers. The model is compared with the traditional employment model.

---

[2] Software development requires high levels of expertise. In commercial software development, complexity can create significant barriers for new developers and users. As a software program grows, changes and becomes more complex, only a few people who have been actively involved in its development over time might fully understand the software architecture and be able contribute additional code. (Fichman and Kemerer 1997)

**Motivation for Study**

From a pragmatic standpoint, the motivations of open source and gated source software developers are of interest to academics and practitioners for three primary reasons. First, individual developers are voluntarily doing work that appears similar to work that commercial firms would pay them to do. This challenges commonly-held beliefs and theories in economics and management studies. An understanding of motivation is fundamental to human resource management and the design of organizations. A firm that provides developers with highly valued benefits might be able to create a more attractive work environment, thereby increasing the firm's ability to attract and retain a highly skilled workforce.

Second, the presence of open source communities affects the strategic direction taken by software firms (Valloppillil 1998). Understanding the motivations and behaviors of community participants brings us a step closer to understanding where such communities are likely to appear and be successful. Because open source software competes with proprietary software, such information may be strategically useful to commercial firms deciding whether or not to enter, remain in, or exit markets where communities currently exist or are likely to enter.

Third, open source software development is an example of a new (to academics and some practitioners) and interesting model of innovation and product development. The assumption that for-profit firms and entrepreneurially-minded individuals are the primary sources of innovation runs deep. However, as a key assumption, it limits our exploration and understanding of the innovation process as it occurs outside of firms. Individuals outside of firms innovate frequently and are the source of important innovations in several product categories studied to date. We currently know little about the complex motivations and social behaviors of individuals involved in this process. Open source and gated software development are representative of a "community-based" model for generating products and innovations. Such open and voluntary collaborative

technology development processes are not unique to software; they exist in other industries and product areas.

The paper is structured as follows. The existing literature on motivations for participation in voluntary product development communities is discussed in Section 4.2. The research sample and inductive methods are described in Section 4.3. Research findings are reported Section 4.4 and discussed in Section 4.5. A model of participation that focuses on the effects of evolving motives and its implications is discussed in Section 4.6. Section 4.7 concludes.

## Section 4.2: Literature Review

### 4.2.1 Community-Based Innovation & Product Development: A Different Mode of Organizing

Much research has focused on the provision of resources in product development organizations, inter- and intra-firm product development-related communications, innovations made by user firms and component suppliers rather than manufacturers of the product (Enos 1962; Knight 1963; Freeman 1968; Allen 1977; von Hippel 1988; Ancona and Caldwell 1992; Brown and Eisenhardt 1995). In contrast, the academic literature to date virtually ignores product development and innovation activities occurring outside the boundaries and structures of firms.

A handful of existing studies, as well as autobiographical accounts and histories of technology, indicate that innovative activity routinely occurs outside of firm in a variety of product areas, including software, electronic components, automobiles, and sporting equipment (Salus 1994; Kline and Pinch 1996; Franz 1999; Raymond 1999; Luthje 2000; Haring 2002; Franke and Shah 2003). Such activities have existed for many years and can produce novel products competitive with those produced by firms (Pugh, Johnson et al. 1991; Salus 1994; Franz 1999; Raymond 1999; Shah 2000; von Hippel 2001).

While some innovators and product developers outside of firms may have worked alone, it appears that many worked jointly with others, either face-to-face, or by communicating through newsletters, journals, magazines, and occasional meetings or conferences, and/or electronic or on-line communication (Salus 1994; Franz 1999; Raymond 1999; Shah 2000; Haring 2002). Recent research has found that individuals working within such communities tend to share information freely with other community participants and often receive assistance from others (Franke and Shah 2003). In addition, they solve problems and design prototypes using information within their existing knowledge domain and often engage in customizing the product for their own use and for their friends (Luthje, Herstatt et al. 2002; Franke and von Hippel 2003).

"Communities" are composed of loosely-affiliated individuals with common interests[3]. They are characterized by a lack of formal coordination and the free flow of information. These characteristics allow for rich information and feedback and the matching of problems with individuals who possess the ideas and means to solve them. Due to the varied skills and needs of individuals involved, user communities are generally well-equipped to identify and solve a wide range of design problems.

### 4.2.2 Motivations for Participation, Creating Code, and Sharing Information

The relatively recent success of two open source software products, the Linux operating system and Apache web server, has piqued academic and general interest in the open source phenomenon. Not surprisingly, a number of explanations for the phenomenon have been suggested. This section reviews existing explanations for why individuals participate in open source software development communities and for two sets of specific actions undertaken by participants: (1) creating and analyzing information (e.g. writing software code to address an issue) and (2) sharing that information with the community.

---

[3] The concept of community employed in this study is an emic one. The websites of both software development projects studied prominently emphasize the importance of "community" in developing high quality software and use the term descriptively: "The **community** factor is an essential component … We value communities more than software…"

### *Motives for Participation*

What do people participate in the creation of open source software? Many competing and contentious theories of motivation have been put forward to answer this question. For example, some argue that political ideology and anti-corporate sentiment fuels open source development. Others argue for the preeminence of enjoyment and creativity, satisfaction of user needs, building a reputation within the community, a need for affiliation, a desire to create or maintain an identity, or training. Table 4-1 provides an overview of existing explanations for participation. Note that an individual may possess multiple motivations and different individuals may possess different motives that influence the choices they make (Jencks, Perman et al. 1988; Jensen and Meckling 1994; Stern 1999; Benkler 2002).

Two relatively large academic survey-based studies on general motives for participation in free and open source software development have been conducted (Niedner, Hertel et al.; Ghosh, Glott et al. 2002). These studies find that motives related to needs for software, improvement in skills, and enjoyment are especially important. Table 4-2 summarizes the key findings of the Niedner, Hertel, et al study. The findings of the Ghosh, Glott, et al study are similar.

**TABLE 4-1:**
**CATEGORIES OF MOTIVATIONS FOR PARTICIPATING IN INNOVATION & PRODUCT DEVELOPMENT COMMUNITIES**

| Motive | Example | Selected References |
|---|---|---|
| **Need For Product** | Participating in order to create, customize, or improve a product or feature | Raymond 1999<br>Kuan 2000<br>Franke & von Hippel 2003 |
| **Enjoyment, Desire to Create and Improve** | Participating because one enjoys it; finds creating or improving software creative and interesting | Weizenbaum 1976<br>Bailyn 1983<br>Gelernter 1998<br>Linus Torvalds 1998<br>Gabriel & Goldman 2001 |
| **Reputation and Status Within the Community** | Participating in order to build or maintain reputation or status within the community | Rheingold 1993<br>Raymond 1999<br>Gabriel & Goldman 2001 |
| **Affiliation** | Participating in order to socialize or spend time with like-minded individuals | Haring 2002<br>Raymond 1999 |
| **Identity** | Participating in order to reinforce or build a desired self-image | Haring 2002 |
| **Values, Ideology** | Participating to promote specific ideals, e.g. the free-software philosophy | Raymond 1999<br>Gabriel & Goldman 2000<br>Stallman 2001 |
| **Training: Learning, Reputation Outside The Community, Career Concerns** | Participating to improve one's skills, with the belief that such improvement will lead to a better job or promotion | Raymond 1999<br>Lakhani & von Hippel 2000<br>Lerner & Tirole 2000<br>Lancashire 2001<br>Hann 2002 |

**TABLE 4-2:**
**REASONS FOR PARTICIPATION AS RATED BY LINUX PARTICIPANTS**

| Item | Importance<br>1 = Very Unimportant; 5 = Very Important |
|---|---|
| Facilitating my daily work due to better software | 4.6 |
| Improving my programming skills | 4.6 |
| Having fun while programming | 4.6 |
| Personal exchange with other software developers | 4.2 |
| Career advantages due to experience gained in Linux projects | 3.7 |
| Gaining a reputation as an experienced programmer inside the Linux community | 3.5 |
| Time loss due to my involvement in Linux | 2.6 |
| Lack of payment for my work on Linux | 2.2 |

*Source: University of Kiel Survey Study (Niedner, Hertel et al.)*

***Motives for Creating & Analyzing Code***

In this section, motives for a specific "work" activity – creating and analyzing code – are discussed. The distinction between intrinsic and extrinsic motivation might be helpful in understanding the type and quantity of an individual's participation with regards to creating and analyzing code. When a person adopts an intrinsic motivational orientation, his primary focus is on the rewards inherent in engagement with the activity; the activity is approached as an "end in itself" (Kruglanski 1975). In contrast, when a person adopts an extrinsic motivational orientation, her primary focus is on rewards that are mediated by but not part of the target activity; the activity is approached as a "means to an end" (Kruglanski 1975). Intrinsically rewarding activities are associated with characteristics such as novelty, entertainment value, satisfaction of curiosity, and opportunities to experience and attain mastery of a particular topic or skill. In contrast, extrinsically rewarding activities are associated with a desire to quickly, predictably, simply, and easily complete the task.

An individual's motivational orientation towards a task may be malleable; for example, when individuals are given external rewards for engaging in an otherwise intrinsically motivating activities, the individuals are less likely to engage in that activity in the absence of the reward and perceive the activity as less intrinsically motivating (Deci 1971). From this perspective, developers may view software development in the context of open source as intrinsically motivating, whereas software development at work – with deadlines, guidance from project managers, etc – may be viewed as something one does primarily to earn a paycheck. The products of work done under different motivational orientations may differ in quantity and quality along a variety of dimensions, including creativity (Amabile 1983; Amabile 1985).

Individuals seek out engaging and intrinsically motivating activities from rock-climbing to chess-playing (Cziksentmihalyi 1996). Engineers are no different, except that the activities they choose may be similar to those they do at work. Haring reports that many amateur radio enthusiasts who built and improved their own equipment as a hobby were engineers – some even worked for radio and electronics manufacturers

(Haring 2002). Bailyn & Lynch (1983) report that engineers with a "puzzle orientation" who do not find adequate stimulation and challenge at their day jobs tend to engage in a variety of technical activities (e.g. improving their cars, building models) in their discretionary time. At least some engineers who do not find "enough" satisfaction in their day jobs have hobbies (like open source software development) that use their engineering skills.

***Motives for Sharing Code or Other Information***

One might expect individuals (or firms) to guard any information that they believe is valuable. However, we observe participants of innovation and product development communities sharing information regularly, and often even commenting on the usefulness of a particular change or addition. It is possible that these individuals see no reason not to share the information, especially if they do not plan to commercialize or otherwise profit from it. Even in that case, however, individuals bear the costs of communicating the information they possess. In this section, three sets of explanations for why information is shared are discussed.

From a sociological perspective, Kollock discusses four motivations that might lead an individual to contribute valuable information to an online group: (1) the expectation that one will receive useful help and information in return, that is, an anticipation of reciprocity, (2) a desire for prestige or status within the group, (3) an increased sense of self-efficacy derived from the act of contributing, and (4) a felt attachment or commitment to the group or project that leads the individual to act in the group's interest (Kollock 1999).

From an economic perspective, it may be beneficial for an individual to reveal innovation or product-related information, if sharing (1) induces improvements by others, (2) sets an advantageous standard, (3) does not assist competitors, and (4) leads to gains from reciprocity and reputation effects (Harhoff, Henkel et al. 2000). Lerner and Tirole focus on career concerns as the primary factor which may lead an individual to share

information and code with others, e.g. a need to learn new skills or a desire to benefit from reputation earned within the software community (Lerner and Tirole 2002).

From a psychological perspective, perceptions of fairness may weigh heavily into an individual's decision to work with others. Work in psychology and behavioral game theory suggests that an individual's willingness to cooperate is highly contingent on assessments of the expected level of cooperation from others. For example, the following factors have been found to mediate an individual's willingness to contribute in public goods experiments: (Dawes and Thaler 1988; Rabin 1998, p. 21-24) (1) beliefs or observations regarding how much others have or are contributing, (2) pre-decision communication, and (3) judgments regarding the behaviors, motivations, and intentions of those who might benefit. In one set of experiments, levels of contribution declined when the same set of players engaged in repeated trials of a public goods experiment with the same group. However when players were told that they would play the game several more times with a *different* group, contributions at the start of the second round went back up to virtually the same rate observed on the initial trial of the first round (Dawes and Thaler 1988, description of experiments conducted by others). The exact reason for this is not known, but it appears that individual beliefs regarding what constitutes fair behavior influence actions in situations where little is known about the opposing party, regardless of prior experience with different actors.

Kahneman, Knetsch, and Thaler (1986a, 1986b) also find experimental evidence for three "rules" of fairness: (1) individuals care about being treated fairly and treating others fairly, (2) individuals are willing to resist the actions of unfair parties even at a positive cost to themselves, (3) individuals have systematic and implicit rules that specify which actions are considered fair and unfair. From where do these "rules of fairness" arise? Evolutionary psychologists and biologists continue to gather evidence supporting the idea that specific cognitive adaptations in the human mind exist for reasoning about social contracts. These adaptations are finally honed for "cheater detection" and can assess the costs and benefits of a social contract from the perspective of different parties (Barkow, Cosmides et al. 1992, p. 206).

## Section 4.3: Method, Setting & Data

This research project uses an inductive research approach, based on the principles of grounded theory building, to better understand the motives and actions of individuals involved in voluntary software development communities. The main questions we sought to answer were: Why do participants voluntarily work with and contribute to community-based product development projects? How are their work efforts and products coordinated? Grounded theory building is particularly useful in situations where the phenomenon does not fit existing categories or is not readily explained by existing theories (Glaser and Strauss 1967). Grounded theory building is a well-accepted methodology among qualitative researchers in sociology, and is used in strategy and technology and innovation management as well (Dougherty 2002). It has three distinct requirements: theoretical sampling, making constant comparisons, and using a coding paradigm to ensure conceptual development (Strauss 1987; King, Keohane et al. 1994).

The primary data collection method used in this study was interviews with participants of voluntary software development communities. As a result, the unit of analysis reported on here is the individual software developer. However, in the interviews, particular attention was paid to the individuals' perception of the community as a social system - the community's goals, "norms and values, local status symbols, social cleavages, and how all these shaped peoples' lives and the community's functioning." Our assumption was that each software developer was choosing how to invest her time. The choices she made, we conjectured, would depend greatly on the social system surrounding her; and her choices would affect that system as well (Coleman 1994, pg. 32-34).

### 4.3.1 Factors Guiding Choice of Setting

As mentioned earlier, there are two categories of voluntary community-based software development: open source development and gated source development. Both types of communities seek to attract volunteer developers and allow developers to view the source code, however different governance principles apply in each community. I

chose to study participation in both types of communities in order to see if differences affected the type, quality, and reasons for participation.

The key distinctions between the open and gated source communities studied are as follows. In the open source community, anyone can download, use, modify, and distribute the code. In the gated source community, the corporate owner of the code specifies the terms of the license and how decisions will be made in the community. Only those who have agreed to a license with the corporate owner can download, use, or modify the code. In addition, in gated communities, some types of use require the payment of a royalty to the corporate owner of the code and there are generally restrictions on code distributions. See the Appendix for more information.

Interviews with six open source software experts and internet searches were conducted in order to gain a better understanding of software development and identify potential communities to study. The two communities studied here were chosen due to their similarities on several dimensions. By selecting for similarity on these dimensions, the behaviors of individuals within these communities could be compared as in a natural experiment.

The two communities are similar in the following ways. Both have a large number of participants; are well known within the software development community; are of roughly the same age; and use the same programming language, thus the pool of developers familiar with or wanting to learn the language would be the same size. The codebases produced by these communities are most often used by developers to develop other software or software applications. The initial code for both communities was developed by the same corporation. Finally, both communities serve as "umbrella" organizations for several projects. Representative projects within each community were chosen for study with the help of community experts.

### 4.3.2 Data Collection & Analysis

Data from four primary sources informed the study. Multiple sources of information enabled triangulation and validation of theoretical constructs.

(1) **Mailing-Lists:** I read all postings to both the project-specific and general mailing lists for both communities studied for a three month period preceding the interviews. Over 2000 messages were posted during this time period. Reading them allowed me to gain familiarity with the types, quantity, and content of discussion and interaction – and the contributions and roles of different individuals.

(2) **Interviews:** I conducted over 60 semi-structured one-on-one interviews with volunteer community participants between December 2001 and March 2002. The majority of interviews were conducted by telephone and tape recorded. Interviews lasted from 30 to 150 minutes.

Interview questions focused on gaining an understanding of early and current participation; methods of processing mailing list information; type of work done; background; current employment context; and the decision-making, governance and ownership practices used within the project. At the conclusion of the interviews, participants were asked to comment on explanations for participation cited in the literature that they had not brought up.

Informants were chosen to maximize variance on the following dimensions related to participation:

(a) *Length of participation.* Defined as length of time since first post to a project mailing list: less than 2 months, more than 2 months (data on the exact length of time was also collected).

(b) *Current frequency of participation*. Measured by number of posts made to mailing lists in the preceding one-month time-period: 1-2, 3-10, more than 10.

(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, "committer" (a participant who has been granted "write access" to the source code repository. Applies to the open source community only), or employee of corporate owner (applies to the gated source community only)?

Information on these dimensions was gathered from observed behavior on project mailing lists (a, b, c) or other project-related documentation (d).

(3) **Conference Observation:** I spent 3-days (approximately 28 hours) observing and meeting with attendees at a technical conference focused on the gated source project.

(4) **On-line Project Documentation:** In addition, I analyzed other project data archived on the Internet, which detailed project interactions and developments. Project data collected from online archives includes project descriptions, charters, bylaws, meeting minutes, mailing list archives, and one informal survey of the "committers" belonging to the open source community. The survey focused on motivations for participation. It was distributed by email and the results were published on the Internet approximately 9 months after this study's interviews were completed.

Interview notes and other qualitative data were coded and analyzed in accordance with grounded theory principles. The variance in the data collected through theoretical sampling facilitated the process of constant comparison and the development of codes and constructs.

Interviewees were guaranteed anonymity to promote candid responses. For this reason, the names of the individuals interviewed, as well as the names of the open source and gated source communities and projects studied are withheld.

### 4.3.3 Operational Definitions

The following operational definitions were used for data categorization and analysis:

*Participant:* posted at least one message to the mailing list[4].

*Short-term participant:* a participant for less than two months at the time of interview.

*Long-term participant:* a participant for more than two months at the time of interview.

*Committer:* a participant who has been granted "write access" to the source code repository (only applicable to participants of open source communities).

## Section 4.4: Findings

The main findings of the study are briefly summarized here and elaborated upon in the following sections. The interviews revealed that short and long-term participants in both the open and gated source communities initially became involved because they needed to use the software. Among those who remained in the open source community, a subset began to work on the project as a hobby. In contrast, virtually all long-term gated source participants continued to be motivated by a need to use the software.

Both interview and archival data indicate that hobbyists are critical to the long-term viability and sustainability of open source software code. They take on tasks that might otherwise go undone and generally make decisions based on what would be best for the code and the community using the code. In interviews, they express a desire to maintain the simplicity, elegance, and modularity of the code.

Section 4.4.1 describes the characteristics of individuals in the sample. Section 4.4.2 discusses communication within the communities. Reasons for creating & analyzing information are discussed in Section 4.4.3 and reasons for contributing information – fairness, feedback, building a better mousetrap – are discussed in Section

[4] It is assumed that many people download and use the software, yet never post a message on the mailing list. Very little is known about such individuals. For this group, the software serves its purpose without additional work, the individual is able to do this work on their own, or the individual receives assistance from sources outside the on-line community. Such users of open source software cannot be contacted directly since open source communities do not keep a record of who downloads the software; such users of the gated software have signed the licensing agreement, however their names are known only to the corporate sponsor.

4.4.4. Formal and informal community rules and processes that support these motives and assist coordination are discussed in Section 4.4.5.

### 4.4.1 Characteristics of Individuals in the Sample

The sample was comprised primarily of college-educated males in their mid-twenties to mid-thirties; only one female was interviewed. The majority held full-time jobs as software developers or engineers. A few were employed by software consulting firms and very few were independent contractors. The characteristics of interviewees in the gated and open source communities were similar and are reported in Table 4-3. Note that the sample of 45 referred to in this and all other tables includes (1) only those developers not employed by the sponsor of the corporate community, and (2) the theoretically sampled interviewees only (not interviews with experts or other individuals).

The amount of time the participants spent doing project-related work varied greatly both across individuals and over time within individuals. At one extreme were participants who downloaded the software and encountered a very minor problem when deploying it. At the other extreme were several participants who spent 2-3 hours per day (occasionally more), 4-6 days per week on project related activities, many of these individuals have been active for well over 6 months.

**TABLE 4-3:**
**SAMPLE CHARACTERISTICS**

| | **Short-Term Participants** | **Long-Term Participants** | |
|---|---|---|---|
| | *(n = 10)* | *General Participants (n = 28)* | *Committers (n = 7)* |
| | | | |
| **Age Range** | 24 to 39 | 24 to 40 [a] | 23 to 37 [b] |
| | | | |
| **Highest Education Level** | | | |
| High School | 1 | 2 | 0 |
| BS/MS | 9 | 23 | 7 |
| ABD/PhD | 0 | 3 | 0 |
| | | | |
| **Employed when Interviewed (Students)** | 10 | 24 (2) | 7 |
| | | | |
| **Married or Committed** | 7 | 18 | 4 |
| | | | |
| **Have Children** | 4 | 10 | 3 |
| | | | |

n = 45; [a] 5 unknown, n = 23; [b] 2 unknown, n = 5

### 4.4.2 Modes of Communication

A defining characteristic of the open source project studied was that virtually all interchange between participants took place in a public forum. The primary means of communication between participants was via the community mailing lists. Thus documentation of virtually all communication was available on the public archives[5]. Note however that a few of those interviewed asked for the assistance or opinions of co-workers at their place of employment before checking public archives for information or sending an email to the community.

The pattern was quite different within the gated source community. Although a great deal of communication took place on the mailing lists, a larger fraction of interviewees report private and off-line communication (1) with the corporate sponsor about the software code or licensing terms, (2) with co-workers, and even (3) with other community participants. Thus it appears that much more work was done off-line in the gated project studied.

### 4.4.3 Reasons for Creating & Analyzing Information Evolve Over Time

As indicated above, open and gated source participants are generally motivated by need, although a subset of long-term open source participants are motivated by the fun and challenge that come with writing and understanding software code. Individuals in the latter set generally joined the community because they had a need, but, over time, began participating for the fun of it. Many of these individuals now consider open source software development a hobby. A corresponding transition from need-based to fun-based participation was not observed in the gated source community.

---

[5] There were very few other exceptions to this rule. Occasionally a committer reported emailing other committers to let them know that they would not be contributing for a while due to work or personal reasons. Very occasionally a committer reported emailing another developer to ask if they would be interested in becoming a committer before making the suggestion publicly. A few community participants report meeting at conferences or local user group meetings (either broadcast to the group prior to the conference or occasionally at random when names are recognized). A few community participants mentioned conducting very specific technical discussions over IRC. The results of these technical discussions were always posted on the project mailing lists.

***Initial Involvement: A Need for the Software***

When describing their initial involvement with open or gated source software, virtually all interviewees spoke of needing to use the software for work-related purposes (Table 4-4). Their needs and the extent to which they had to manipulate the software to meet those needs varied widely, e.g. some had questions about how to use the software in a particular context, while others needed to write a new feature for their own purposes.

> *"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, USA, age 29

Many made the choice to use open source software, rather than commercially-available software specifically because they could view and change the code to best fit their own needs.

Several gated source participants expressed reluctance to use gated software, stating that they would have preferred the software to be open source. The licensing mechanism made it difficult for them to get permission from their managers and/or corporate legal departments to use the software. Many said that the technical capabilities of the gated software were unparalleled and necessary to solve the problems they were working on. Most undertook comprehensive searches for other software options before choosing the gated software.

**TABLE 4-4:**
**NEED FOR SOFTWARE DRIVES**
**INITIAL VOLUNTEER INVOLVEMENT**

| **Reason For Initial Involvement** | **Short-Term Participants (Open Source/Gated)** | **Long-Term Participants** | |
|---|---|---|---|
| | | *General Participants (Open Source/ Gated)* | *Committers (Open Source)* |
| | | | |
| **Need for Software** | 9 (7 / 2) | 26 (11 / 15) | 7 |
| | | | |
| **Other** | 1 (0 / 1) | 2 (1 / 1) | 0 |
| | | | |
| n = 45 | | | |

***Long-term Involvement: Volunteers Behave Differently In Open Source & Gated Communities***

When asked to describe the reasons for their ongoing participation, virtually all long-term gated source project participants continued to focus on their need for the software. They mentioned specific needs for new functionality or for up-to-date information on changes in the software that might affect them (Table 4-5). The reasons for ongoing participation reported by long-term open source participants were strikingly different (Table 4-6): fewer than half reported continuing their participation because of need.

**TABLE 4-5:**
**MOTIVES IN GATED COMMUNITIES**
**DO NOT APPEAR TO CHANGE OVER TIME**

| **Long-Term Gated Source Community Volunteers** | | |
|---|---|---|
| ***Motive*** | ***Initial Involvement*** | ***Current Involvement*** |
| | | |
| **Need Software for Own Use** | 15 | 14 |
| | | |
| **Other** | 1 | 2 |
| | | |
| n = 16; chi squared = 0.61 | | |

**TABLE 4-6:**
**FOR THOSE WHY STAY IN OS COMMUNITIES,**
**MOTIVES EVOLVE OVER TIME**

| **Long-Term Open Source Community Volunteers** | | |
|---|---|---|
| ***Motive*** | ***Initial Involvement*** | ***Current Involvement*** |
| | | |
| **Need Software for Own Use** | 18 | 7 |
| | | |
| **Other** | 1 | 12 |
| | | |
| n = 19; chi squared = 3.76; $p < 0.001$ | | |

***"Needless" Long-Term Involvement: Enjoyment***

What, if not need, motivates more than half of all long-term open source members interviewed? When describing why they chose to engage in certain tasks, this subset of participants spoke of their open source work as a fun and challenging hobby-like activity (Table 4-7).

> *"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

In fact, the website of a related project even includes a list of haikus written by developers, many focusing on software development itself. The following example evokes the implicit tension between a seductive hobby and "legitimate" (known to the boss) work.

> *Time, too much have you*
> *major geeks these people are*
> *boss know you do this? :)*

**TABLE 4-7:**
**LONG-TERM PARTICIPANTS WITHOUT OWN NEED FOR SOFTWARE DESCRIBE THEMSELVES AS MOTIVATED BY CHALLENGE & FUN**

| Primary Motivation for Creating & Improving | Gated Source Community Participants | Open Source Community Participants | |
|---|---|---|---|
| | | *General Participants* | *Committers* |
| | | | |
| **Challenge and fun** | 1 | 6 | 5 |
| | | | |
| **Reputation outside community** | 1 | 1 | 0 |
| | | | |
| **Other (learning, obligation, reciprocity, ideology, etc.)** | 0 | 0 | 0 |
| | | | |
| n = 14 | | | |

These developers focus on the importance of keeping the code elegant and simple, as well as backward compatible[6]. Because they are also users of the software, they have experience with bad code and an inherent interest in maintaining the quality and backwards compatibility of the code. They see the project as something very useful and meaningful. These preferences guide their choice of tasks, methods of addressing tasks, and choices regarding which suggested features to add or develop and which to exclude from the source code.

> *"Now that I am a committer I am more concerned about backwards compatibility and other issues; I am more conservative."*
>
> - Long-term participant, open source community, Australia

[6] Backward compatibility means that a software system can successfully use interfaces and data from earlier versions of the system.

This group contributes to the development of the software in many ways. Importantly, interview data (from members of this group and others) suggests that these participants take care of a great deal of "maintenance work" such as committing code, designing new releases, etc. In contrast, newcomers or infrequent participants identify problems and contribute many of the new ideas.

Note that many committers who participate as a hobby do commit code contributed by others and patch bugs. However they do so when they have the time and desire. Some of these committers are far more active than others. Although bugs are generally addressed quickly, backlogs of code embodying new features or promising greater efficiency often build up. Oftentimes the back-logs are 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. Committers who reported need (not fun) as their primary motive generally did not commit code unless (a) they had written it themselves, and (2) it pertained to the portions of the code in which they were most interested. A long-term participant motivated primarily by his own need for the software reports:

> *"Even now I don't incorporate code into the tree unless I write it… I didn't write the code… I do make minor revisions… it's worth it now that I have access."*
>
> - Long-term participant, open source community, USA

Participants reported choosing work tasks that were related to the portions of the code they had previously worked on. Several reported that their knowledge of the content and structure of the code accumulated over time, allowing them to undertake more difficult challenges, including those that required an understanding of more than one area (module) of the code. Learning appeared to be a by-product, that is, a reward,

resulting from engaging in challenging and fun work, not a primary driver of the work itself[7] [8].

The types of tasks and work undertaken vary, however virtually all interviewees stressed the fact that the vast majority of the work was done by choice:

> *"No one makes you do anything here, if you see a need or something that would be fun to work on, you take on the responsibility and do it… you let others know what you are doing, obviously."*
>
> – Long-term participant, open source community, USA

> *"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'll leave …"*
>
> – Long-term participant, open source community, France

---

[7] A parallel can be seen in crossword puzzles. Many people enjoy doing cross-word puzzles and are likely to learn some new words in the process; however, they do not do the puzzles to learn new words. Others may engage in solving crossword puzzles in order to learn new words, but they often do not enjoy it in the same way nor will they keep at it for long periods of time.

[8] 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 *Iliad*. Most participants pointed out that they had come into the project with the required technical skills and all stated that they learned about the structure of the open source code as they worked. Some reported sharpening or expanding their skills. Whether a willingness to engage in technical problem solving is innate or partially driven by knowledge of a particular piece of software code is not known and would make an interesting topic for future research.

Many open source developers – even those who attained committer status – leave their projects. They are not expected to remain on the project indefinitely and exit is understood to be a normal part of the process. It is common to use and have worked on several open source projects, although most reported that they can only dedicate a great deal of effort to a single project at any one time. The character and behaviors of the other people involved in a project influence participation decisions:

> *"At some point, the participation decision isn't 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?"*
>
> - Short-term participant, open source community, USA

**4.4.4 Reasons for Contributing Information**

Even if one has made an alteration to the software or understands it well, the costs of contributing this information are positive and often relatively high. Time and effort are required to communicate, e.g. code might have to be cleaned up; thought must be put into what is useful to others and what might be particular to your own needs; comments and explanations must be composed. Despite these costs, many individuals contribute information – either code they have created or knowledge - to the community. The reasons for contributing work vary by individual and over time, but several patterns can be observed. In this section, the role of fairness, the desire for assistance from others, and the desire for feedback on one's work are discussed.

Several of the individuals interviewed reported that they did not contribute all or part of their work product. The reasons given varied and included: 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.

***Fairness***

Several short-term participants and a few long-term participants who spoke about the importance of fairness and feelings of obligation in motivating their decision to share information and code with the community. Within the open source project, individuals citing this as a reason were primarily interacting on the "user" mailing list (not the developer mailing list). They reported that "others helped me, so I should help them" and "this is what is done in the community." After contributing, several interviewees who cited this reason removed themselves from the mailing lists and did not keep up with software-related developments[9]. For these individuals, it does not appear that contribution is tied to seeking reputation-related recognition or future improvements on the contributed code. Both of the two individuals in this category did not know if their code contribution had been accepted and made part of the code ("committed"); they were using the version of the code they had initially downloaded and then altered.

> *"My questions were answered, the product worked, I took my name off the mailing list."*
>
> - Short-term participant, open source community, USA

***Need For Assistance When Building a Better Mousetrap***

Both short- and long-term participants reported interest in finding better solutions than the ones currently in place. 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 may be helpful to themselves and others, (3) communicate that they had an important need that was worth spending their own time solving and might thus be worthy of the attention of others.

[9] The reasons for this were not explored, but might include that they felt they had contributed enough back, contributed most of what they thought they knew, lost interest, or observed that others – at that time – were not interested in the same issues.

In the process of scanning through the mailing lists for topics related to their own, they often came across questions from others that they knew the answer for and answered them, out of reciprocity or fairness; this pattern of answering questions has been documented in other studies (Lakhani and Hippel 2000).

Only a subset of those who brought up the desire to have an even better solution said that they wanted to get their code incorporated into the source code base so that it would remain in future versions of the code. Several remarked that what was important was having the feature or functionality and that they tried to avoid upgrading to new releases, preferring instead to rely on what they currently had for as long as possible.

***Open Source Projects: Feedback on Creative & Useful Work***

As discussed earlier, many people work on open source projects because they enjoy programming. However, enjoyment of the act of programming does not necessitate that one contribute or even work within a community (one could work alone). Individuals appear to contribute in order to receive feedback upon their work. Feedback comes from both those who use the product and those who participate as a hobby.

Many participants report monitoring the mailing lists for feedback on the portions of the software they have created and for ideas regarding what interesting problems exist that they might want to work on. In the process, they encounter questions and requests for assistance. They are very selective in choosing what to read and whether or not to respond. Most provide assistance only when (1) others might not be able to answer the question, and (2) when it looks as if the requestor of information will actually make changes and contribute to the project.

> *"I only take part in developer list design debates that are going to go somewhere… whoever is suggesting has to be asking a "how" question."*
>
> - Long-term participant, open source community, USA

Several interviewees mentioned that they felt an obligation to support software they had written. However that obligation was related to helping others understand the code and tinker with it themselves, and not to provide services or customize the software for individuals[10].

Recognition from one's peers is one type of feedback and may tie into the desire to be creative and engage in challenging tasks.

> *"...creative programmers want to associate with one another: only their peers are able to truly appreciate their art. Part of this is that programmers want to earn respect by showing others their talents. But it's also important that people want to share the beauty of what they have found. This sharing is another act that helps build community and friendship."*
>
> - Expert, USA

Attaining "committer" status is an additional form of feedback in the open source community. This lets the individual know that their contributions are valued by existing committers. It can also serve as a signal that lets others in the community know what committers value, thereby establishing norms. It is possible that such recognition affects contribution decisions; however interviewees generally regarded being made a committer as a welcome "pat on the back" rather than something they diligently worked to attain. Several committers mentioned cases where nominees asked not to be made committers. They also stated that many committers stopped working on the project soon after their individual needs were fulfilled and that most of those who worked on the project for fun left after a year or so.

[10] This makes sense as an action that is fair and promoted future feedback: if users are ignored, they are likely to find other software to use and feedback will decline. If you are having fun as you create the code and enjoy the feedback, answering questions is the least you can do. The mechanism for doing this is partially institutionalized: when asking a question, the requestor is expected to have searched past archives and looked at the code, thus when they email their question to the community they often email the mailing list AND cc any individuals known to have knowledge of the topic. This decreases the search costs for providers of information and increases the likelihood that the requestor will receive a quick response.

*"Most 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."*

- Long-term participant, open source community, Australia

***Gated Project: Character of Contributions***

As reported in an earlier section, volunteers did improve the gated product to satisfy their own needs, but rarely worked for fun or enjoyment. As a result, much necessary work does not get done by volunteers, even though that work would arguably be fun, challenging, and exciting.

Two factors, fairness and a desire for feedback, strongly affect work and contribution within the gated project[11]. More specifically, restrictions on use, modification, and distribution keep volunteers from taking on work not related to own need, because "it is only fair to be able to use what one developed." In addition, restrictions on who can license and use the code and for what purposes limit overall use of the code and thus feedback.

*"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]*

[11] Alternate explanations might be that the gated source software is inherently less challenging or that volunteers are relying on employees of the corporate owner to do work, however (1) the gated source project studied is generally regarded by software developers to be more exciting and revolutionary than the open source project studied. 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 owner do as much work as possible, especially the more mundane work, this does not explain why volunteers are not seeking out the work that is (self) defined as fun and exciting.

*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, USA

*"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 largest licensees in mind… it's political… I just have a small business… why should they care what I need?"*

- Long-term participant, gated source community, USA

*"In an open source community, no 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, USA

### 4.4.5 Achieving Community "Coordination"

How are the work efforts and products of those who contribute coordinated? The term "coordination" carries the connotation of a hierarchical (or flat) structure in which work goals are defined and tasks are carved off and assigned to individuals. This type of coordination was not observed in the communities studied, except by employees of the gated source community sponsor. Instead, participants identified tasks which they felt needed to be done and then worked on providing a solution. The mechanisms and processes that facilitated this type of "loose" coordination in the communities studied are discussed *briefly* below. The processes described tend to "conserve" three resources central to the community: participant time, space on mailing lists, and participant motivation.

#### *Selected Aspects of Task Coordination*

How do individuals select tasks and create boundaries around those tasks? How are tasks carried out? This section describes the role of feedback, answering questions, and joint problem solving on task definition and execution. Task coordination also involves the process by which changes and additions to be made to the source code are selected; this process requires detailed research and is not addressed here.

*Feedback*

Feedback from others can be used to identify areas of the code that need improvement, identify new features that users desire, or provide input on planned or completed work. There was an informal rule among developers in the open source community to "ask first and then do" so that ideas would be exposed to peer review before an individual spent his or her time executing the idea.

Community development exposes one person's idea to critique and review by anyone interested in the issue. Those who offer opinions or ideas must be prepared to defend, and possibly alter, them[12]. It has been argued that this open and diverse - due to

the heterogeneity of interested participants with respect to issues such as computing environment, design skills, knowledge of the code base, and needs – peer review process contributes to the high quality of community developed code (Raymond 1999; Mockus, Fielding et al. 2000).

Two hobbyist developers who dedicated considerable amounts of time to the project pointed out that feedback sometimes comes directly from the code:

> *"Sometimes you work on an area and you notice that the code is getting more and more complicated – hard to understand and comprehend - say because many small changes have been made. There comes a time when you need to start from scratch and rewrite the code with all the new functionality in mind. Otherwise you look at it and get confused and everyone is less likely to be able to understand and improve that area."*
>
> - Long-term participant, open source community, Germany

[12] In contrast, tactics based on power or authority might be used to influence development decisions within a firm or joint venture. Such tactics are unlikely to fare well in many open source environments and/or may inhibit community growth. Corporations interested in working with communities – and their employees – are especially aware of the difference:

> *"As a firm, we can't dictate how everything works in the project and in fact, you have to go along with decisions that may not be best for you… it's a very different development environment and I constantly have to remind my boss and colleagues of that … We can't dictate who will do what and then. If we want a change, we have to contribute the manpower to make it happen."*
>
> - Long-term contributor (employed to participate), open source project, USA

*Providing Information*

Two interesting patterns regarding information provision by participants were observed. First, participants skim mailing list headings and choose which posts to read. Most report read messages that pertain to past or current work, and occasionally look at a topic because it looks interesting or has attracted the attention of many in the community.

> *"The worst thing someone can do is write "question" or "help" in the subject line. You've irritated everyone before they even look at your question – but a few people will take a look and try to help, if you are lucky."*
>
> - Long-term participant, open source community, USA

Second, when choosing if to respond to a question, many short-term developers respond to virtually any question they think they can answer. In contrast, the majority of more experienced participants report (1) answering only questions that they think others can not easily or accurately address, and (2) responding only when it appears that the information requestor will do work and contribute something to the community.

> *"I answer questions that others probably can't answer… why spend time answering a question that someone else could take care of?"*
>
> - Long-term participant, open source community, USA

> *"At first I offered many opinions and suggestions. Then I learned to only respond to "how" questions."*
>
> - Long-term participant, open source community, Germany

*Joint Problem Solving*

Some problems are identified, solved, and executed by a single person. Other problems are solved jointly by several people. Sending an email suggesting an idea and solution concepts attracts the interest and assistance of others.

> *"The more people you can attract to an issue, the better. You'll get a good solution and that we'll incorporate it into the source code more quickly because we can see that there's interest and that it's been looked over carefully."*
>
> - Long-term participant, open source community, Germany

***Selected Aspects of Behavioral Coordination***

Behavioral coordination involves the enforcement of community rules and practices. The community relies on developer participation and actions that might decrease participation are punished or reprimanded (see Table 4-8 for a summary of behaviors generally deemed unfair or inappropriate behavior by communities versus firms). The community's methods for providing punishments are limited, being generally confined to discussion, instruction, or embarrassment on mailing lists and/or not assisting a participant in the future. Four areas where behavioral coordination came up repeatedly are discussed here: software use, attainment of committer status, requests for assistance, and tone of feedback[13].

*Software Use*

Use of the software is encouraged, but distribution of software without referencing its origins receives harsh criticism. Distribution of software without referencing its origins limits feedback – and the potential for subsequent improvement –

[13] This is not an exhaustive list.

and also appears to violate fairness norms. In one case, a developer in the open source community studied noticed that a consultant for his employer had taken a significant amount of code from the open source project, but had neglected to cite the project. He announced the consultant's name to the open source community, as well as to his employer, colleagues, and other software developers he knew.

*Attainment of Committer Status*

As mentioned before, in the open source community studied, only those with "commit privileges" may alter the source code. Access to this privilege can be granted only by a vote from existing committers. Three positive votes are required for privileges to be granted, a single negative vote means that privileges are denied. The voting process is conservative and is likely to act in favor of preserving existing norms and practices.

*Requests for Assistance*

The projects studied had only one or two mailing lists associated with them. The positive benefit of having just one or two primary mailing list is that everyone "congregates" in one place, thereby increasing the possibility that a problem will be paired with an individual who has the knowledge to address it. The downside is that all requests for information and assistance are sent via these mailing lists, thereby increasing the volume of posts potential information providers must sort through. In order to make efficient use of the system, four informal rules for requesting information exist: (1) search the mailing list archives and FAQs before asking a question. Requests for information that is documented in the FAQs (frequently asked questions) or archives often receive a response suggesting that the individual to check these sources and not waste the time of others; (2) devise an appropriate and descriptive subject line header; (3) phrase request in an appropriate manner[14]; (4) send your request to the community, but copy individuals who may have expertise in the area on the email. This way, the

[14] What constitutes acceptable behavior may differ by community. Even within the same umbrella project, participants of different subprojects behaved differently, e.g. several participants of a more supportive and low-key sub-project pointed out the more aggressive and competitive behaviors exhibited within another sub-project.

requester may get advice even if the information experts do not skim the mailing list that day.

*Tone of Feedback*

Positive feedback through use and critical feedback are both encouraged. Demotivating feedback (insults, blame, demands for improvements) is often met with contempt and result in a "do it yourself" messages from *several* developers, as illustrated by the following exchange.

**Portion of Email message:**

*"Once again, it's all chaos at [name of umbrella project]. I don't understand why the folks with the capabilities and responsibilities to handle infrastructure issues and requests don't take five minutes to set up a [tracking] category for these things. Instead, it is left to endless e-mails, no tracking, nothing gets done..."*

**Portion of Response 1:**

*"Please take note of the tone of your email. I understand your frustration, this is a volunteer organization, and while the volunteer system works well for software, it is not always great for infrastructure.*

*Regardless, this is where we are. You're asking for someone to do you a "favor" and fix [problem name]. Do you feel your tone does it in a way that will motivate them to do so?*

*Let us now address the [tracking] issue. I've noted my preference for this as well, however, I do not currently have the bandwidth necessary to drive the discussion towards that. You, however, are empowered to do so! Join the infrastructure list and start the discussion. I would suggest that in order to be effective, you describe the problem (calmly), describe the solution, the best alternative and why you feel [solution name] would be*

*appropriate. Inevitably someone will argue against it or just "for" the status quo. Through persistence, offering to help manage it, etc, your issue may be addressed.*

*Complaining about [name of umbrella project] on this mail list will probably just peeve people off and serve you nothing, although you're certainly welcome to try it.*

*Is this the most judicious use of your (and others') time?*

*A man once said "You decide".”*

**Response 2:**

*“My response to postings like this is: Thanks for volunteering! =)”*

**TABLE 4-8:**
**SOME COLLECTIVE ACTION ISSUES AND RESPONSES**

| | Employment Model in Most Firms | Voluntary Participation Model in Communities |
|---|---|---|
| | | |
| **What Constitutes Free-Riding?** | • Not producing expected work product | • Expecting others to undertake additional work on your behalf<br>• Wasting the time of others by asking questions that have been asked before or for information that will not be acted upon<br>• Not reporting problems experienced while using the software, esp. if such problems are likely to affect others |
| | | |
| **Responses to Free-Riding** | • Individual-level incentives (may decrease cooperation between individuals and result in less information sharing and innovation)<br>• Monitoring<br>• Termination of employment | • Specifically state that product support is not provided and should not be expected<br>• Public or private statement explaining acceptable and non-acceptable behavior<br>• Being ignored<br>• Removal of individual's ability to post messages to mailing list<br>• Automatic bug messages sent to community when problems encountered |
| | | |
| **What Constitutes Theft?** | • Using firm resources for personal benefit<br>• Appropriation of proprietary information or equipment | • Not referencing the community as the source of all or part of the code |
| | | |
| **Responses to Theft** | • Legal action<br>• Termination of employment | • Harsh public statement condemning action and reporting the individual's name<br>• Legal ownership of code by the collective, which may threaten to enforce property rights (never done before) |

## Section 4.5: Discussion

This section examines two issues in detail: (1) the ways in which the concept of fairness influences community participation and (2) the process by which contributions are assessed, selected, and incorporated into the code.

### 4.5.1 Fairness: Responses to Contribution & Governance

Fairness appears to strongly influence community participation in (at least) two ways. First, individuals must choose whether or not to contribute their knowledge. Second, individuals must choose the extent of their community involvement. These issues are discussed in the following two paragraphs.

Many short-term participants and a few long-term participants said they contributed primarily because they thought it was the appropriate thing to do within the community. The observation of activity on the mailing lists, as well as help they themselves had received in the past appear to have promoted this perception. Two processes may be at work: (a) A social facilitation effect whereby observing the contributions of others leads one to contribute as well. Kollock discusses the impact of this process in the context of open source software development and the recruitment of volunteers for public service work (wiring public school classrooms) over the Internet (Kollock 1999); (b) Repaying the community for help one had previously received. In the context of software development communities, such repayment, or reciprocity, has two interesting characteristics. First, it takes the form of generalized exchange, because individuals contribute what they know to whoever needs it, not just the person(s) who provided them with assistance. This is likely to occur when knowledge is being exchanged, since there is no guarantee that the assistance you can provide will be of use to the person who provided you with assistance. Second, because of the public nature of the exchange, contributions are shared with the entire community. This increases the information set from which an individual in the community can draw upon and helps the

community as a whole progress. Such public exchange is not limited to the software community or the Internet, and is likely to be widespread among amateur hobbyists[15].

Community participants must also make decisions regarding the extent of their involvement within a particular community. Community (institutional) rules and processes deeply impact an individual's decision whether or not to create or "play with" the software beyond what their own needs require. The developers interviewed expected continued access to software they had helped develop and the ability to use that software in whatever context they wished. The licensing structure of the open source software project met both of these requirements. The gated software did not. The effect was three-fold. First, many gated project participants reported that they used the gated software only after other options had been exhausted and found unsatisfactory[16]. This decreases the size of the community and thus the volume of activity that takes place within it. Second, many gated source community participants – especially those not affiliated with large corporations on good terms with the corporate owner - felt uneasy with the software and how software-related decisions would be made. They felt a need to make sure their voices were being heard and their interests were being represented. Third, individuals rarely worked on the gated-source project to satisfy anything but their most pressing needs. This decreases both the volume and scope of work undertaken by volunteers[17].

[15] Skateboarders and other sports enthusiasts have created websites directing others on how to do the latest tricks or improve their equipment. Newsletters and magazines were created to facilitate information exchange between innovative amateur automobile enthusiasts in the early 1900s and innovative sports enthusiasts in the late 1900s (Franz 1999; Shah 2000). Ham radio operators in the mid-1900s, responsible for several commercially important technological developments in electronic components, communicated via newsletters and radio waves (Haring 2002).

[16] They found the licensing structure unfair, especially since they would be required to do a considerable amount of the work necessary to make the software useable. Moreover, the licensing terms made corporate legal departments uneasy and added another barrier to use that had to be overcome. Gated participants who worked for large corporations, especially those with an existing relationship with the corporate owner of the gated software, had an advantage in that they were more readily able to negotiate licensing terms for commercial use.

[17] In a sense, developers who might have worked on the software for fun are not participating at cost to themselves (Kahneman, Knetsch et al. 1986b).

If the software is very much needed, that is it provides truly unique and useful functionality for some users (enough so that those users are willing to contribute to its development) and a corporate owner is willing to hire employees to do some of this work, the effect of the first and third issues may be partially overcome. Decreased participation and general mistrust of the fairness of the process, however, are likely to continue and to affect both the character and volume of participation[18]. In this paper, one element of the licensing structure – open source vs. gated source – was examined, however other elements of the community governance structure may also play a role in affecting the extent of an individual's involvement.

### 4.5.2 How Do Communities Turn "Ad Hoc" Contributions Into High Quality Products?

Many processes contribute to the creation of high-quality products in communities. Three are of critical interest: the ability to build upon and use the work of others, feedback, and the presence of hobbyists. The open and often public nature of exchanges, as well as electronic or paper documentation of past exchanges, allows everyone access to information and build upon it. Free use and distribution of the product is critical. When use and distribution are restricted, as in the gated source community studied, individuals generally did not create or contribute anything more than that required for their own use. Positive and negative feedback serve, respectively, to encourage creation and contribution, and guide improvements and future work.

The presence of hobbyists who care about product design is critical to the community. Their presence increases the likelihood that the code will remain of high quality on a number of dimensions (e.g. modularity, compactness, simplicity), thereby making the code accessible to new users who seek to understand and improve parts of the code. In contrast, individuals who are motivated only by need might create the new

[18] The community would be without the "need"-neutral voices of those who participate as a hobby and prevent the code from becoming too large or too messy or incorporating too many features. It is unlikely that such individuals are "selected out" of the community from the onset since they are likely to have the same distribution of software needs as others; they probably enter the community and only work to satisfy their own needs, and spend their discretionary "hobby" time on other projects. There is some evidence of this in the sample.

feature and "tack it on" to a part of 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[19].

## Section 4.6: Model

A model that explains technical progress in user communities must address why individual developers choose to create within and contribute to the community, as well as how developers overcome technical and knowledge barriers and how contributions are transformed into high-quality products, with a sustainable architecture. The model outlined below was derived inductively.

### 4.6.1 Need Drives Initial Participation

The primary driver of initial participation – both creating and contributing – is the need for changes or alterations to the product. Participants initially search for existing solutions to their problems, and only if no solution is found do they create one on their own or with the assistance of others[20]. Because a need exists, many individuals do not wait for others to solve the problem.

Some of these individuals contribute the solution back to the community, others do not. A desire for improvements on their work leads some to contribute. These individuals continue to monitor the mailing lists for discussion regarding their needs and often help others working in the same area. Unless an individual believes she has the "best" solution, she is likely to contribute in order to benefit from potential improvements to the idea by others. Some report that they contribute back because "it seems like the right thing to do." They often contribute; however, they often also exit the community

[19] 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 "add 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.

[20] Assistance may be requested from community participants (in open source communities, communication occurs primarily via mailing lists) or others who the individual knows (e.g. co-workers at their day job).

soon after their needs are met[21]. The amount of assistance received from others appears to increase the likelihood of contributing something back.

### 4.6.2 Reasons for Participation Change Over Time

An individual's reasons for participating may change over time (Figure 4-1). Need may initially lead the developer to the community. However as the individual gathers information and learns more about the code in order to solve his or her problem, he or she may find other problems of technical interest in the same or other areas of the code[22]. Gelerntner and others discuss the beauty of code and the psychological pleasures that can be derived from interacting with it (Weizenbaum 1976; Gelernter 1998).

> *"... almost any project will give a developer that "feel good" feeling when he has users and he feels he is doing something worthwhile. I really don't think you need all that much "quid pro quo" in programming – most of the good programmers do programming not because they expect to get paid or get adulation from the public, but because it is fun to program."*
>
> - Linus Torvalds (Torvalds 1998)

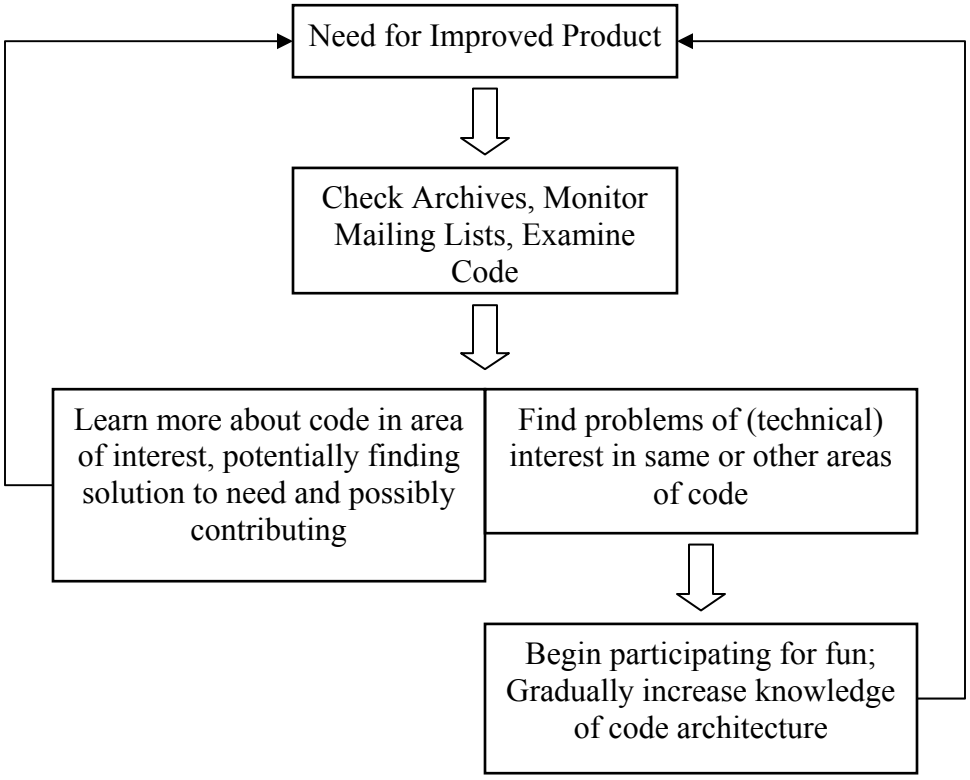
In the case of open source participation, developers find such problems as they scan through the mailing lists looking for information related to current interests. A particular subject line might catch their interest and prompt them to read the message. This may result in investigation of the related code and participation in related work. Over time, the individual learns more about the structure and architecture of the project's code base, increasing the individual's knowledge base and ability to manipulate and improve the code.

---

[21] In a sense, these developers have decided that the existing solution is "enough" for their problem – or lack the time or desire to seek an improved solution.

[22] This is in line with the empirical observation that open source software developers often specialize in one or two areas of the code. (von Krough and Spath 2003)

**FIGURE 4-1:**
**EVOLUTION OF PRIMARY MOTIVES OVER TIME**



### 4.6.3 Enjoyment & Personal Satisfaction Drive Many Long-Term Developers

A subset of those who participate for an extended period of time, do so primarily because they enjoy programming and open source projects are a venue in which they can choose the programming activities they engage in and get feedback on their work (Table 4-9). Participation is often viewed as a hobby. Contribution of the code is necessary to obtain feedback.

Feedback enhances intrinsic motivation in certain situations[23]. Positive feedback in these communities occurs via two channels: use and occasional commentary on the code by others (generally those with an interest in code structure and architecture). Feedback often creates additional work, primarily responding to bug reports and assisting others who are working on issues related to the code you wrote. Developers appear receptive to this type of work, since it often creates interesting puzzles for them to solve; in fact, many report monitoring the mailing lists for discussions related to what they wrote.

> *"So the large user-base has actually been a larger bonus than the developer base, although both are obviously needed to create the system… I simply had no idea what features people would want to have, and if I had continued to do Linux on my own it would have been a much less interesting and complete system… the thing is that a Linux user has a very hard time not giving things back. He doesn't have to give anything back actively: it is feedback to just know that some person uses Linux for a certain application domain. And usually even a very silent user tells a lot more than that – just by the type of questions he posts to newsgroups and mailing lists."*
>
> - Linus Torvalds *(Torvalds 1998)*

[23] Cognitive evaluation theory argues that intrinsic motivation is based on feelings of competence and self-determination - and that external constraints, such as contingent rewards, often influence these feelings based on their "informational" and "controlling" characteristics (Deci 1975). External informational characteristics provide feedback about competence and can either enhance or detract from intrinsic motivation depending on content. Substantial empirical evidence supporting cognitive evaluation theory exists (Deci 1971; Lepper and Greene 1978; Boggiano and Pittman 1992).

These developers are often interested in technical problems, although their interests vary widely. Because they have already satisfied their own immediate software needs, they have an understanding of some area of the product and make further modifications based on the needs and ideas of others, or partake in work that they deem necessary to improve the software's capabilities or design (architecture). Attention to technical detail is important to this, often small, subset of developers. They are generally interested in striking a balance between usability (and number of standard features) and keeping the code simple, elegant, and easy to understand. The latter allows developers to independently make whatever changes they require and precludes the need for voluminous documentation.

These same developers appear to avoid "managing" the project. Most posts containing an introduction and a request to be assigned a task are either ignored or the requester is told to monitor the mailing lists[24]. Individuals interested primarily in reputation or status building must first find a niche that interests them and where they have the skills to be useful. This can be a time-consuming and frustrating exercise. While some or many may attempt it, the developer who participates and engages in the community without a specific need is rare. This is in stark contrast to the management process in most firms where a project manager assigns tasks (Table 4-10). The practice of turning away such requests and making potential participants find and solve their own problems effectively creates a screening mechanism whereby only those with skill, time, and desire to program become a short- or long-term part of the community.

[24] In a sense, the developers appear to shun any work that does not focus on the code itself and their preferences for such "pure" work appear to be in line with Abbott's conception of professional purity: "by professional purity I mean the ability to exclude non-professional issues or irrelevant professional issues from practice… the highest status professionals are those who deal with issues predigested and predefined by a number of colleagues. These colleagues have removed human complexity and difficulty to leave a problem at least professionally defined, although possibly still difficult to solve (Abbott 1981, p. 823)."

**TABLE 4-9:**
**LINKAGE BETWEEN CREATION, CONTRIBUTION, AND KNOWLEDGE OF CODE STRUCTURE**

| Reason to Create | Reason to Contribute | Relative Level of Individual Participation | Relative Number of Participants | Knowledge of Code Structure ("Learning") |
|---|---|---|---|---|
| | | | | |
| **Need** | *Future product improvements* | Varies, depends on need | High | Primarily in area of initial problem, may expand slightly |
| | | | | |
| | *Fairness* | Low | High | Limited to area of initial problem |
| | | | | |
| **Fun, Enjoyment** | *Feedback* | High | Low | Initially area of initial problem, gradually broadens |
| | | | | |
| **Reputation** | *Reputation-Related Benefits* | Low[a] | Very low | Difficult to begin; difficult to find a "niche" |
| | | | | |

[a] Two primary exceptions exist: some individuals interested in building a reputation choose to do documentation work; others may find an initial area to work on based on matching their existing knowledge of some area of software development with a question or suggestion posed by another individual.

**TABLE 4-10:**
**PARTICIPATION MODELS IN FIRMS VS. COMMUNITIES**

| | Employment Model in Most Firms | Voluntary Participation Model in Communities | |
|---|---|---|---|
| | | | |
| **Who can Participate?** | • Employees are screened and selected | • Anyone | |
| | | | |
| **Primary Motives for Initial Involvement** | • Money and employment benefits | • Need for product or product feature | |
| | | | |
| **Primary Motives for Extended Involvement** | • Money, employment benefits, power | • Need for product or feature | • Fun, Challenge |
| | | ⇩ | ⇩ |
| **Tasks Undertaken** | • Most often assigned and monitored by managers or owners | • Related to own needs | • Varied: can include maintenance of product and organization |
| | | • Chosen by participants and undertaken voluntarily | |
| | | | |
| **Primary Exchange Concepts** | • Employee-firm (owner, manager): economic exchange<br>• Between employees of similar position: social exchange | • Social exchange<br>• One to many (or public) communication whenever possible (individual – community and individual-individual both occur simultaneously on mailing lists) | |
| | | | |
| **Authority and Control** | • Owners and managers | • Distributed among individuals<br>• Some communities have formal heads - most try to assert as little control as possible | |

**Section 4.7: Conclusion**

Hobbyist hackers who work for free are not new to the computer industry, as illustrated in the following quote:

> *"In the late 1960s, just outside Seattle, a group of teenagers met after school each day and biked to a local company. As it closed for the day and its employees began heading home, the boys were just getting started. They routinely thought of themselves as the firm's unofficial night shift, and in fact they routinely worked until long after dark, pounding on the keys of the company's DEC computer and gorging on carry-out pizza and soft drinks. The two leaders of the group [Paul Allen and Bill Gates] were considered a little odd by their classmates. They were "computer nuts," completely absorbed in the technology. All the boys worked for free… Computer Center Corporation, which they called "C Cubed" let them come in to find errors in the DEC computer's language… as long as C Cubed could show that DEC's program had bugs (errors that caused the programs to malfunction or "crash"), the firm didn't have to pay DEC for using the computer* (Freiberger and Swaine 2000)*."*

Although there are many reasons why an individual might participate in a product development community, a need for the software and enjoyment lead the list. Hobbyist developers are critical to the functioning of a voluntary software community. They take care of many maintenance needs and oversee the overall code design from a relatively neutral perspective. A large set of users and developers, each with their own set of needs and agenda, are necessary to generate a variety of ideas and solutions from which those that offer valuable functionality can be chosen. Communication between developers and users of the software is critical and a software community is likely to flourish only when anyone can enter and when each participant feels empowered to do as much or as little as he or she desires. Limiting participants' ability to alter and use the code through any one of a variety of formal or informal mechanisms is likely to decrease participation and may either drive hobbyists away or lead them to change the nature of their activities.

There are several limitations to be considered when interpreting and using the results of this study. The study takes an in-depth look at two of the many software development communities that exist today. Studies of additional communities, large and small, are needed. Additional empirical is needed to further investigate the linkages between an individual's motivations and types of contributions he or she makes, especially among the many participants who each contribute only a little, but in aggregate contribute a great deal. Finally, the software developed in both projects mentioned here is generally used by developers working within corporations. Software developers participating in other projects might have different individual characteristics (e.g. age, education level) and motivations for creating and contributing. For example one might expect learning or ideology to be a more common "cause" among college students or observe that groups of developers motivated primarily by ideology are most likely to create software that substitutes for equivalent proprietary software[25].

Interesting areas for future research involving community-based innovation and product development include: *Problem Domain:* In what problem domains are we likely to see or not see communities emerge? *Governance:* What governance mechanisms are likely to increase or decrease active participation? *Community Creation:* How are communities started? What variables impact their early growth? *Status and time:* How will the status structures created naturally affect community evolution over time? What are the ramifications of individual efforts to achieve high-status? *Selection of Problems and Solution Concepts:* Once problems have been identified and potential solutions contributed, how do committers choose what to incorporate and what not to incorporate into the code? Design issues and feedback from the community regarding what is useful and not useful play a role, but more detailed work is needed. *Corporate Participation:* How will communities react as corporations begin to see and use communities for their own strategic ends? What will be the effect on product development?

[25] One would still expect that a substantial portion of the work done in such communities would still be done by people who enjoy developing software. After all, there are many other ways to express ideology or build reputation besides spending hours writing software code.

## Appendix

### What Is Open Source Software Development?

There are many definitions of what constitutes open source. The basic idea is simple: by making the source code for a piece of software available to all, any programmer can modify the software to better suit his or her needs and redistribute the improved version to others[26]. By working together, a community of both users and developers can improve the functionality and quality of the software. To be open source requires that anyone can get and modify the source code, and that they can freely distribute any derived works they create from it. The different licenses have various wrinkles on whether modifications must also be made into open source or if they can be kept proprietary (Gabriel and Goldman 2001, entire paragraph).

### How Are Open and Gated Source Development Different From Other Methods of Producing Software?

Software development methods can roughly be characterized as either proprietary or community-based (Figure 4-2). Proprietary software is often owned and developed by a corporation. The corporation sells binary code to users of the software (consumers), who then install and run the binary code on their computers[27]. Binary code is made up of a series of 0's and 1's and can be understood by a computer, but not by an individual. If a user finds a problem with the software or has an idea for a new product feature, he or she must request the change from the company who sold them the software.

Community-based software can either be owned by a firm or by another actor. However, the source code, not the binary code, is distributed to users. The source code is software code written in a particular programming language. Anyone with knowledge of the language can understand and alter the software. The user converts the source code into binary code and runs it on his or her computer. Should the user want to make a

---

[26] The formal Open Source Definition can be found at: http://www.opensource.org/docs/definition.html

[27] Individual purchasers of software as well as firms who contract with software design companies for specific products often receive binary code as an end-product. In the latter case, contractual arrangements are often made to hold the source code in "escrow" should the purchasing company need it in certain situations.

change to the software, he or she can change the source code, convert it to binary, and then run it on the computer. The user can then tell others about the modifications he or she has made. While only company software engineers have the ability to change proprietary code, many individuals have the ability to change community based code.

There are two general approaches to community-based software development: gated development and open source development (Figure 4-3). Both seek to attract volunteer developers, but take different approaches to licensing and community governance. Gated development is the more restrictive of the two, and limitations on who can view, use, modify, and distribute the code may be specified by the owner of the code.

**Historical Roots of Open Source Software Development**

Open source and community based software development are not new concepts, they are as or nearly as old as the history of computing. The historical antecedents of what we now call "open source" can be seen in the development histories of the early IBM 360 Systems, the Unix operating system, and much of the software developed in university labs in the 1960s and 1970s[28] (Bashe, Johnson et al. 1986; Salus 1994). Using a community to develop and share software made sense as a practical matter and does not appear to have been promoted as a philosophical matter.

In 1984 Richard Stallman formed the free software foundation and started the GNU project (http://www.gnu.org). He did so in response to seeing the collapse of the software-sharing community at the MIT Artificial Intelligence Lab as many left to join companies making proprietary software. Stallman coined the term *free software* to express his philosophy that programmers should be allowed access to the source code so they could modify it to suit their needs.

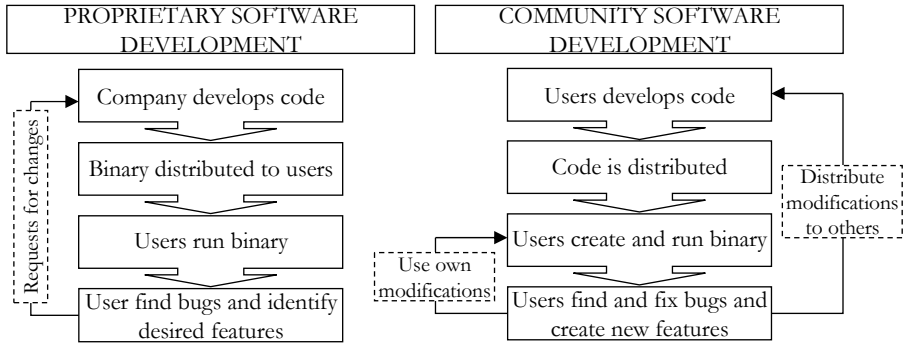
> "I consider that the golden rule requires that if I like a program I must share it with other people who like it. I cannot in good conscience sign a nondisclosure agreement or a software license agreement."
>
> - Richard Stallman, 1983, http://www.gnu.org/gnu/initial-announcement.html

[28] There is a long history of software code written in the OS tradition; much of which directly or indirectly benefited firms and certainly benefited society as a whole. For example, early open source communities were critical to the development of early IBM software (Pugh, Johnson et al. 1991). IBM managers were conscious of and strategic in their decision to encourage customers of their hardware products to share software and software related information. This practice effectively increased the value of IBM hardware to the customer, while requiring less time and investment by IBM than if they had tried to investigate and satisfy customer needs independently. In contrast, early prototypes of the Unix operating system were a stepchild project at Bell Labs (Salus 1994). However, project engineers promoted Unix outside of Bell Labs and encouraged others (outside of Bell Labs) to continue development (Salus 1994); allowing others the ability to contribute to the project fueled its development and kept it alive.

He developed the GNU General Public License (GPL) to assure that he would always be able to see and modify the source code for any software he wrote, along with the modifications made by anyone else who might work on it (Gabriel and Goldman 2001).

Further details on the history of open source are available on-line and in books like *Open Sources: Voices from the Open Source Revolution* (edited by Chris DiBona, Sam Ockman and Mark Stone) and *Rebel Code: Inside Linux and the Open Source Revolution* by Glyn Moody.

## FIGURE 4-2: PROPRIETARY VERSUS COMMUNITY SOFTWARE DEVELOPMENT




- Scheduled releases
- Work done and decisions made by paid employees and project managers

- Frequent releases
- Extensive involvement by many

## FIGURE 4-3: TWO APPROACHES TO COMMUNITY SOFTWARE DEVELOPMENT

| | **Gated** | **Open Source** |
|---|---|---|
| Who can use? | Those who agree to license; commercial use may require royalty payment | Anyone |
| Who can modify? | Only licensees | Anyone |
| With who can you share? | Only other licensees | Anyone |
| Who owns? | "Sponsor" retains rights to code (and improvements) | "Owned" by the collective |
| Who manages and makes decisions? | "Sponsor" stipulates and delegates project decisions | Users or "benign dictator" |
| Example licenses | SCSL | GPL, BSD, ASL |

*Both seek to attract volunteer software developers*

## BIBLIOGRAPHY


Abbott, A. (1981). "Status and Status Strain in the Professions." American Journal of Sociology 86(4): 819-835.

Allen, T. J. (1977). Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information Within the R&D Organization. Cambridge (MA): MIT Press.

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

Amabile, T. M. (1985). "Motivation and Creativity: Effects of Motivational Orientation on Creative Writers." Journal of Personality and Social Psychology 49: 393-397.

Ancona, D. G. and D. F. Caldwell (1992). "Bridging the Boundary: External Activity and Performance in Organizational Teams." Administrative Science Quarterly 37: 634-65.

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

Bashe, C. J., L. R. Johnson, J. H. Palmer and E. W. Pugh (1986). IBM's Early Computers. 3rd edition. Cambridge, MA: MIT Press.

Benkler, Y. (2002). "Coase's Penguin, or, Linux and the Nature of the Firm." Yale Law Journal 112(3).

Brown, S. L. and K. M. Eisenhardt (1995). "Product Development: Past Research, Present Findings and Future Directions." Academy of Management Review 20: 343-348.

Cziksentmihalyi, M. (1996). Creativity: Flow and the Psychology of Discovery and Invention. New York: Harper Collins.

Dawes, R. M. and R. H. Thaler (1988). "Anomalies: Cooperation." Journal of Economic Perspectives 2(3): 187-197.

Dougherty, D. (2002). Building Grounded Theory: Some Principles and Practices. Companion to Organizations. J. A. C. Baum (J. A. C. Baum): Blackwell Publishers: 849-867.

Enos, J. L. (1962). Petroleum Progress and Profits: A History of Process Innovation. Cambridge, MA: MIT Press.

Fichman, R. G. and D. F. Kemerer (1997). "The Assimilation of Software Process Innovations." Management Science 43(10): 1345-1363.

Franke, N. and S. Shah (2003). "How Communities Support Innovative Activities: An Exploration of Assistance and Sharing Among End-Users." Research Policy.

Franke, N. and E. von Hippel (2003). "Satisfying Heterogeneous User Needs Via Innovation Toolkits: The Case of Apache Security Software". MIT Sloan School Working Paper #4341-02. Cambridge, MA.

Franz, K. (1999). Narrating Automobility: Travelers, Tinkerers, and Technological Authority in the Twentieth Century. Department of American Civilization. Providence, Rhode Island: Brown University: 291.

Frederick Brooks, J. (1975). The Mythical Man-Month: Essays on Software Engineering. Reading, MA: Addison Wesley.

Freeman, C. (1968). "Chemical Process Plant: Innovation and the World Market." National Institute Economic Review 45(August): 2957.

Freiberger, P. and M. Swaine (2000). Fire in the Valley. 2nd edition. New York: McGraw-Hill.

Gabriel, R. P. and R. Goldman (2001). "Collaborative Development Handbook: How a Company Can Participate in Open Source". California.

Gelernter, D. (1998). Machine Beauty. New York: Basic Books.

Ghosh, R. A., R. Glott, B. Krieger and G. Robles (2002). "Free/Libre and Open Source Software: Survey and Study, Part 4: Survey of Developers". International Institute of Infonomics, University of Maastricht.

Glaser, B. and A. Strauss (1967). The Discovery of Grounded Theory: Strategies for Qualitative Research. New York: Aldine De Gruyter.

Harhoff, D., J. Henkel and E. von Hippel (2000). "Profiting from Voluntary Information Spillovers: How Users Benefit By Freely Revealing Their Innovations". Sloan School Working Paper. Cambridge, MA.

Haring, K. (2002). Technical Identity in the Age of Electronics. History of Science Department. Cambridge, MA: Harvard University.

Jencks, C., L. Perman and L. Rainwater (1988). "What is a Good Job? A New Measure of Labor-Market Success." American Journal of Sociology 93(6): 1322-57.

Jensen, M. and W. Meckling (1994). "The Nature of Man." Journal of Applied Corporate Finance 7(2): 4-19. Reprinted in Jensen, M. C. (1998). Foundations of Organizational Strategy. Cambridge, MA: Harvard University Press.

Kahneman, D., J. Knetsch and R. Thaler (1986b). "Fairness and the Assumptions of Economics." Journal of Business 59(4): S285-S300.

King, G., R. Keohane and S. Verba (1994). Designing Social Inquiry. Princeton: Princeton University Press.

Kline, R. and T. Pinch (1996). "Users as Agents of Technological Change: The Social Construction of the Automobile in the Rural United States." Technology & Culture 37: 763-795.

Knight, K. E. (1963). A Study of Technological Innovation: The Evolution of Digital Computers. Doctoral Dissertation. Pittsburgh, PA: Carnegie Institute of Technology.

Kollock, P. (1999). The Economies of On-line Cooperation: Gifts and Public Goods in Cyberspace. Communities in Cyberspace. M. A. Smith and P. Kollock (M. A. Smith and P. Kollock). London: Routledge.

Lakhani, K. and E. v. Hippel (2000). "How Open Source Software Works: Free User to User Assistance". MIT Sloan School Working Paper. Cambridge, MA.

Lerner, J. and J. Tirole (2002). "The Simple Economics of Open Source." Journal of Industrial Economics 52(June): 197-234.

Luthje, C. (2000). "Characteristics of Innovating Users in a Consumer Goods Field". MIT Sloan School Working Paper. Cambridge, MA.

Luthje, C., C. Herstatt and E. von Hippel (2002). "The Dominant Role of "Local" Information in User Innovation: The Case of Mountain Biking". MIT Sloan School Working Paper. Cambridge, MA.

Mockus, A., R. T. Fielding and J. Herbsleb (2000). A Case Study of Open Source Software Development: The Apache Server. Proceedings of the International Conference on Software Engineering, Limerick, Ireland.

Niedner, S., G. Hertel and S. Hermann (2000). "Motivation in Open Source Projects (http://www.psychologie.uni-kiel.de/linux-study)."

Pugh, E. W., L. R. Johnson and J. H. Palmer (1991). IBM's 360 and Early 370 Systems. Cambridge, MA: MIT Press.

Rabin, M. (1998). "Psychology & Economics." Journal of Economic Literature 36(1): 11-46.

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

Salus, P. (1994). A Quarter Century of Unix. Reading, Massachusetts: Addison-Wesley.

Shah, S. (2000). "Sources and Patterns of Innovation in a Consumer Products Field: Innovations in Sporting Equipment". MIT Sloan School Working Paper. Cambridge, MA.

Stern, S. (1999). "Do Scientists Pay to Be Scientists?" NBER Working Paper. Cambridge, MA.

Strauss, A. (1987). Qualitative Analysis For Social Scientists. New York: Cambridge University Press.

Torvalds, L. (1998). "First Monday Interview with Linus Torvalds: What Motivates Free-Software Developers?" First Monday 3(3).

Valloppillil, V. (1998). "Halloween Documents." Microsoft Corporation http://www.opensource.org/halloween/.

von Hippel, E. (1988). The Sources of Innovation. New York: Oxford University Press.

von Hippel, E. (2001). "Innovation By User Communities." MIT Sloan Management Review 42(4): 82-86.

von Krough, G. and S. Spath (2003). "Community, Joining, and Specialization in Open Source Software: The Case of Freenet." Research Policy.

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