Communications of the Association for Information Systems

Volume 16 Article 37

November 2005

Open Source: Concepts, Benefits, and Challenges

Mohammad AlMarzouq
Clemson University, malmarz@clemson.edu

Li Zheng
Clemson University, liz@clemson.edu

Guang Rong
Clemson University, grong@clemson.edu

Varun Grover
Clemson University, vgrover@uark.edu

Communications of the Association for Information Systems

OPEN SOURCE: CONCEPTS, BENEFITS, AND CHALLENGES

Mohammad AlMarzouq
Li Zheng
Guang Rong
Varun Grover
Department of Management
Clemson University
vgrover@clemson.edu

ABSTRACT

With the emergence of free and open source software (F/OSS) projects (e.g. Linux) as serious contenders to well-established proprietary software, advocates of F/OSS are quick to generalize the superiority of this approach to software development. On the other hand, some well-established software development firms view F/OSS as a threat and vociferously refute the claims of F/OSS advocates. This article represents a tutorial on F/OSS that tries objectively to identify and present open source software's concepts, benefits, and challenges. From our point of view, F/OSS is more than just software. We conceptualize it as an IPO system that consists of the license as the boundary of the system, the community that provides the input, the development process, and the software as the output.

After describing the evolution and definition of F/OSS, we identify three approaches to benefiting from F/OSS that center on (1) the software, (2) the community, and (3) the license respectively. Each approach is fit for a specific situation and provides a unique set of benefits and challenges. We further illustrate our points by refuting common misconceptions associated with F/OSS based upon our conceptual framework.

KEYWORDS: open source, free source, tutorial, system, community, motivation, software development, benefits, challenges, F/OSS, OSS.

I. INTRODUCTION

The terms free [1] and open source[2] refer to software that anyone can freely redistribute, analyze, and modify while complying with certain criteria. With companies looking for new ways for

[1] Definition of Free software: http://www.fsf.org/licensing/essays/free-sw.html
[2] Definition of Open Source: http://www.opensource.org/docs/definition.php

reducing IT spending, free and open source software (F/OSS) emerged as a promising solution for reducing costs. F/OSS proponents claim that it can offer a free alternative to proprietary software with superior quality when it comes to features, reliability, and security [Grantham, 1999, Raymond, 1999c, Surman and Diceman, 2004, Wheeler, 2005]. Proponents also assert that F/OSS can be a means for reducing product development costs by incorporating F/OSS components into a product or outsourcing parts of the product development to F/OSS communities [Grantham, 1999]. On the other hand, opponents of F/OSS solutions argue that open source software (OSS) does not come with a free price tag and the total cost of ownership (TCO) might actually be higher than commercial solutions when other costs such as staff training, customization and implementation are factored in [Hickman, 2004, Surman and Diceman, 2004]. Both Microsoft and Sun conducted studies showing conflicting results about the total cost of ownership of proprietary and F/OSS software [Wheeler, 2005]. The debate regarding whether F/OSS reduces costs is still ongoing. In reality, both F/OSS and proprietary approaches can yield different benefits in specific conditions [Levesque, 2004, Williams et al., 2005].

The focus of this paper is to provide a tutorial on F/OSS. The purpose of this paper is not to propose that the F/OSS approach is better or worse than the proprietary one, but to inform readers who seek to learn more about F/OSS and help managers make a more informed decision about F/OSS solutions.

In this tutorial, we first contextualize F/OSS by reviewing its origin and evolution (Section II). In Section III,we attempt to conceptualize F/OSS and its components . We then identify three approaches through which companies can harness F/OSS. The benefits and challenges associated with each approach are discussed. We conclude by examining commonly held misconceptions about F/OSS in light of the tutorial.

II. ORIGINS AND EVOLUTION OF OPEN SOURCE

The idea of F/OSS is not new. F/OSS dates back to the origins of the computing industry in 1950s. All software was free back then, and most of it was open. People in the field perceived computer hardware and software as highly intertwined. They were not able to see the explicit market value of software. Free and open software dominated the industry until mid-1960s. IBM then unbundled its software and hardware leading to a significant marketplace for software in the 1970s [Glass, 2004].

The new wave of F/OSS discussed here, arguably originated in the late 1990s. It is different in nature from traditional open software. Current F/OSS emphasizes open standards, shared source code, and collaborative development behind the software [O'Reilly, 1999]. The ideals of F/OSS are firmly rooted in the Hacker Ethic, or Hackerdom. Hackers are programmers who enjoy exploring the details of programming systems [Johnson, 2001]. They undertake a project to fulfill constructive goals and their intense creative interests [Johnson, 2001]. The beginnings of the hacker culture dates back to 1961. At that time, MIT's AI Lab was the first software-sharing community. The cultures of programmers in the early years and the following Hackerdom eventually evolved into today's free and open-source cultures [Raymond, 1999a].

The Department of Defense designed and built ARPAnet in the late 1960's as an experiment in digital communication. ARPAnet greatly facilitated the spread of hackerdom. After the initial launch, it quickly grew and started to link hundreds of universities, defense contractors, and research laboratories. It provided a platform for the free exchange of information with unprecedented speed and flexibility. Meanwhile, this network brought together hackers from all over the U.S. in a critical mass. This phenomenon led to the rise of networked groups that enjoyed collaborative effort. Their early efforts led to informal principles and guidelines for distributed software development [Raymond, 1999a].

During the same time, a Bell Labs hacker named Ken Thompson invented UNIX. UNIX was originally licensed to universities for a minimal fee. UNIX resulted in an explosion of creativity and efforts as programmers built on each other's work. Richard Stallman, who was a participant in

MIT's Artificial Intelligence Lab and believed strongly in the Hacker Ethic, created the Free Software Foundation in 1985, an organization that promotes the development and use of free software. Hundreds of programmers created new, freely available versions of all major UNIX utility programs. UNIX, as one of the most widely known shared projects, offered solutions to networking problems and contributed to the ongoing growth of the Internet [Raymond, 1999a].

By the early 1990's, reduced costs and increased performance of personal computers together with the rapid growth of the World Wide Web all contributed to the growth of online development communities. Projects such as Linux and Apache became immensely successful in terms of project contributions. Following the success of these projects, in 1997, Eric Raymond [Raymond, 1999c] presented his seminal piece "The Cathedral and The Bazaar"[3]. He contrasted two different styles of software development, "the cathedral model of the commercial world and the bazaar model of the Linux world." The former is tightly organized and centrally planned. In contrast, Linux development resembles "a great babbling bazaar of differing agendas and approaches." [Raymond, 1999c]. The paper (and the later book version) acted as a catalyst, drawing great attention and discussion to F/OSS. It played a key role in justifying the decision of Netscape's CEO, Jim Barksdale, to release the source code for Netscape Navigator in 1998. Netscape then invited Eric Raymond to help them develop what was to become the Mozilla Public License (MPL)[4] and the Mozilla Organization [Raymond, 2000].[5]

On February 3rd, 1998 a brain storming session was convened in VA research offices in Mountain View, California to discuss the future of this movement. Participants Todd Anderson, Chris Peterson (of the Foresight Institute), John "maddog" Hall and Larry Augustin (both of Linux International), Sam Ockman (of the Silicon Valley Linux User's Group), and Eric Raymond, agreed on the need for a marketing campaign to win the support of Fortune 500 companies to ensure the long term survival of the movement. However, the participants acknowledged that the term free software did more harm than good. They argued that it kept Fortune 500 CIO's away because they associated the term with hostility towards intellectual rights and communism. Further, the term free does not fit very well in the business world. Also, the ambiguity of the term free and the fact that most such software was distributed with no cost, added to the confusion [Raymond, 2000]. Christine Peterson of the Foresight Institute coined the Open Source (OS) label that is synonymous with the bazaar metaphor and the Open Source Initiative organization (OSI) was established by the people present at that meeting[6].

Some opposition to this movement however surfaced. Richard Stallman of the Free Software Foundation (FSF) thought that the term Open Source was not pure enough.[7] Furthermore, the term might be confusing because it emphasizes the access to source code and not the freedom of users. This argument reflects the differing beliefs of FSF and OSI, which promote the same development principles but disagree on the ultimate goal of the movement. FSF decided to keep the free software label to reflect its belief that users should be given the freedom to do whatever it is they wish with their software. FSF is vocal in expressing the user's right for this freedom. OSI, although might agree with these principals, chose to promote this freedom in a more subtle manner. They might not express the user's right for this freedom, but they promote it as a means of producing better software and trying to persuade the corporate world to buy into this concept[8]. Because of this difference, we refer to the group of software that adheres to the OSI and FSF principles as Free and Open Source Software (F/OSS).

---

[3] An update in 2000 is available on the web at http://www.catb.org/~esr/ writings/cathedral-bazaar/hacker-revenge/

[4] F/OSS licenses are explained in Section III

[5] http://www.mozilla.org

[6] http://www.opensource.org/docs/history.html

[7] http://www.catb.org/~esr/open-source.html

[8] For more information on the difference between OSI and FSF consult:
OSI perspective: http://opensource.org/advocacy/free-notfree.php
FSF perspective: http://www.fsf.org/licensing/essays/free-software-for-freedom.html

F/OSS gained further credibility when giants in the IT industry, such as Apple, IBM, and Sun, started adopting F/OSS solutions in the following years [West, 2003]. Google, NASA, and many others now choose to deploy F/OSS.[9] F/OSS (such as Bind, Sendmail, and Apache) and other, proprietary software originally developed by the F/OSS community are the heart of the Internet. Apache, for example, became the most popular web server on the Internet in April 1996. The February 2005 Netcraft Web Server Survey[10] found that more than 68% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined [Feller and Fitzgerald, 2002].

III. THE OPEN SOURCE SYSTEM

F/OSS is more than just software or source code [Hein, 2004]. Four components constitute F/OSS. These are:

1. the license (What signifies a software as F/OSS),
2. the community (why the people get involved),
3. the development process (how F/OSS is conducted), and
4. the software itself (what the product entails).

Collectively, these four components provide a reasonable understanding of F/OSS solutions. Therefore, in order to better evaluate and use F/OSS, it is useful to understand the interplay of the four components as a whole.

To provide a comprehensive understanding of F/OSS, we conceptualized the components and the interplay between them as an Input-Process-Output (IPO) system. For IPO systems, output is but a single part of the whole system. The quality of the output depends on the system as a whole. Therefore, F/OSS is only as good as the process, the community, and the license [Orr, 1998].

The license legitimizes the whole system and signifies it as an F/OSS system. It is the means used to protect the intellectual rights of the contributors and to ensure the sustainability of the system. The community provides all necessary input such as knowledge, skill, time, and effort to produce the final product. The F/OSS development process enables the collaboration of efforts (inputs) to produce the software (output).

Typically, IPO systems involve two orders of feedback loops.

- The first order feedback loop focuses on improving the development process to ensure that the output meets the pre-defined software and quality requirements.
- The second order feedback loop focuses on how to update the pre-defined requirements so the system can cope with changes in external environment.

These two orders of feedbacks enable the F/OSS as an IPO system to work effectively and evolve dynamically. Figure 1 presents an overall picture of F/OSS as an IPO system and illustrates the two orders of feedback. A discussion of each component follows. We start with the license as the overall boundary of the system. We then discuss the I (community), P (development process), and O (software).

[9] http://www.mysql.com/customers/
[10] http://www.netcraft.com

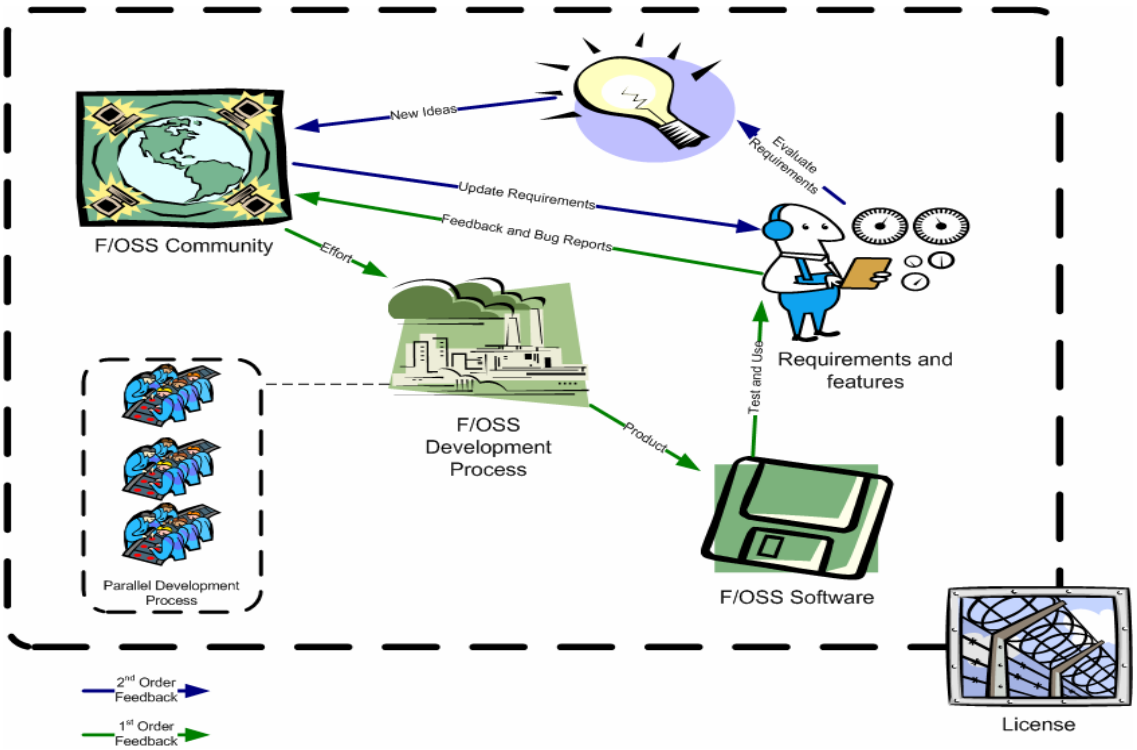

Figure 1: F/OSS as an IPO System

THE LICENSE

A more technical definition of F/OSS (than the one provided earlier) would be software with distribution terms (i.e. license) that comply with well-defined criteria [Bonaccorsi and Rossi, 2003a]. To say that a piece of software is F/OSS indicates that it is subject to the terms of a particular category of licenses [McGowan, 2001]. The license defines the terms by which an individual is to use the software.

F/OSS licenses are very important because they can serve as a governing mechanism that enforces the norms of the F/OSS community, provides motivation for programmers by protecting their efforts from appropriation, and distinguishes F/OSS from proprietary software [Bonaccorsi and Rossi, 2003b].

The numerous F/OSS licenses available all maintain the openness and free distribution of the source code[11]. The difference between the licenses reflects the different philosophies within the F/OSS communities on how to advance the F/OSS projects and the need to deal with issues surrounding a particular piece of software. As discussed in Section II, the Free Software (FSF) movement and the Open Source Initiative Organization (OSI) reflect two different philosophies within F/OSS communities.

> *"The Free Software Movement and the Open Source Movement are two political parties in the same community."* [Richard Stallman as quoted in Wong and Sayo, 2004].

[11] A comprehensive list, of F/OSS licenses, visit: http://www.opensource.org/licenses/ http://www.fsf.org/licensing/licenses/index_html

Both the FSF and OSI offer similar criteria in qualifying licenses as Free Software or Open-Source. Many F/OSS licenses are approved by both. For a license to be qualified as F/OSS by FSF or OSI, it must allow programmers to access, modify, and redistribute the source code. The fundamental difference between the two movements is in their values, their ways of looking at the world. At the heart of FSF is the freedom to cooperate. FSF considers non-free software unethical and argues that users should be free to do what they want with their software. OSI is concerned with the technical values of making powerful and reliable software, and advancing similar principles to FSF by getting the corporate world to buy into their development methodologies. The main focus of OSI is on the development process rather than the underlying moral requirement of maintaining the freedom associated with the software. As FSF states, "Open source is a development methodology; free software is a social movement."[12]. Although the philosophy of the two movements is different, both Open Source and Free Software developers do cooperate on practical problems such as software development, efforts against proprietary software, and software patents.

In summary, the ultimate goal of FSF is to give users the freedom to use software as they wish and are clear about their goal. OSI's ultimate goal is to win the buy-in of the corporate world by promoting the benefits of F/OSS. OSI believes that, as a result, users would obtain the same freedom.

FSF Licenses

It follows that each movement would recommend a different set of licenses. The FSF movement recommends the use of licenses that are similar to the GNU[13] Public License (GPL). The GPL is the most widely used F/OSS license and is the strictest [Lee, 1999]. The GPL protects freedom for all its users and prevents commercial appropriation of their collective effort. The GPL license implements two principles that are, as the FSF sees it, the best means by which to preserve the software freedom of the users.

1. The first principle is referred to as copyleft. The basic idea is that the works derived from the original F/OSS source code base must also remain F/OSS. Any modifications cannot be privatized.
2. The other principle is that the licensed F/OSS cannot be mixed with proprietary source code [Lee, 1999]. This principle is also referred to as GPL compatibility by the FSF[14], since GPL was the first license to implement this principle. Hence the GPL requires any source code linked to the GPL licensed software to be distributed as free software. It effectively prevents people from taking advantage of F/OSS and using the software for their own commercial benefit, by preventing a programmer from establishing copyright or patent rights on the software and the integration with a closed/proprietary source program at the source code level.

OSI Licenses

OSI recommends the use of licenses that fits the company's business model. As discussed in section II, Eric Raymond helped Netscape draft the Mozilla Public License (MPL) in 1998, around the time OSI was established. The license, as a popular alternative to the GPL, does not implement either of the FSF principles. OSI recognizes that the commercial success of the contributors can be a force that can advance F/OSS. The license allows any modifications to be made private and allows mixing proprietary source code with F/OSS source code. It is well suited to programmers who want a greater degree of flexibility in combining F/OSS and proprietary source code. This licensing method is more appropriate for use in a commercial context.

---

[12] FSF official website is at http://www.fsf.org/licensing/essays/free-software-for-freedom.html

[13] GNU stands for GNU's Not Unix. Developed by Richard Stallman and the Free Software Foundation, GNU is a high-quality version of the Unix operating system that is free of charge and freely modifiable by its users. Many GNU applications and utilities are mainstays of the Unix community.

[14] http://www.fsf.org/licensing/licenses/index_html

A common misunderstanding that arises due to the use of the term free and the costless redistribution of source code for most F/OSS projects. Both FSF and OSI agree that software producers can sell copies of their software for a fee. This view does not contradict any of the principles of both organizations. The only issue is whether, after selling the software to the user, the user does or does not have the right to use, modify, and redistribute the software as he see fit without being restricted by the original producer.[15]

In summary, the key difference between various types of F/OSS licenses is the mechanism behind them to enforce the openness of F/OSS [Lee, 1999]. The choice of F/OSS license is affected by the ideological debates within the F/OSS community [Lee, 1999], and these play an important role in protecting and determining the type of participation in the F/OSS community [Hertel et al., 2003].

THE COMMUNITY

The community consists of all the developers and users of the F/OSS. They are dispersed over time and space. Internet technologies and collaborative software offer the means by which the members of the community interact and contribute. The community is the source of all input that goes into the F/OSS system such as source code, requirements, and bug reports [Raymond, 1999c, Wayner, 2000].

Community Building

Open source communities begin when an individual or a group of individuals contribute an initial functional prototype of the software as F/OSS. People then gather around this prototype, with their own reasons and objectives, and work collaboratively to continue developing the software [Raymond, 1999c]. As the software becomes more usable, it attracts more people to the community, provided the software meets the developers' interest. In turn, these new players bring effort and contribution that is geared toward improving the software. A growth cycle starts that feeds both the community and development of the software (Figure 2). This growth cycle creates a network effect that is associated with the size of the community. As the community size grows and becomes more diverse, so do its value and the value of the product (the software). The growth ensures the ongoing survival of the community and further improvement of the product [Raymond, 1999c].

Motivation

Attracting new members to the F/OSS community does not guarantee that these people will contribute. The majority of people in the community are users who do not contribute with code submission [Crowston and Howison, 2005, Krishnamurthy, 2002]. Since the advancement of the project depends on free contributions from the community members, the members should be motivated and technical able to contribute [Bezroukov, 1999, Raymond, 1999c].

Motivation for community participation is as diverse as the people that contribute to F/OSS. It can be segmented into intrinsic and extrinsic motivation. Intrinsic motivation occurs when contribution, in itself, is valued by the individual. This is the case when contribution is enjoyable, intellectually stimulating, or when there

[15] Selling free software: http://www.fsf.org/licensing/essays/selling.html

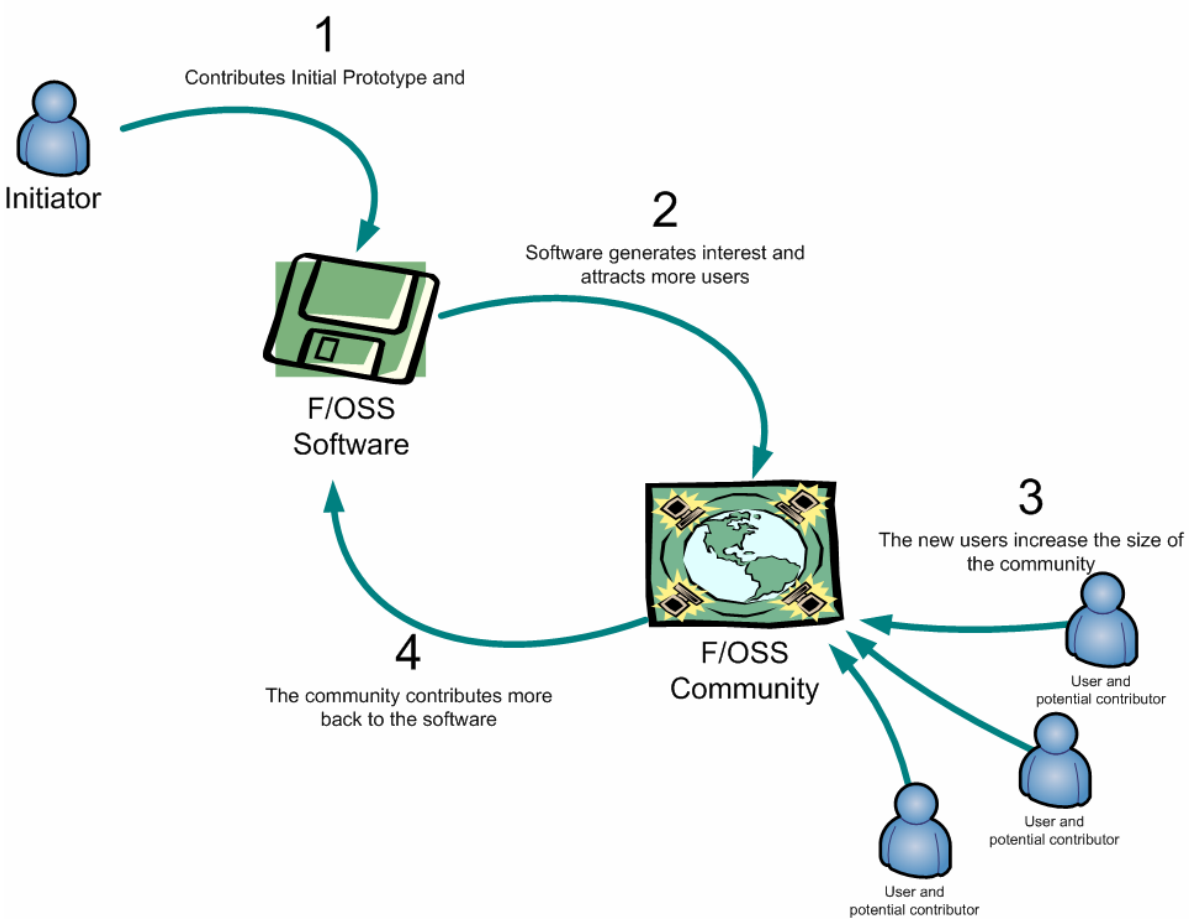

Figure 2. F/OSS Community Initiation and Growth

is a sense of obligation to contribute back to the community. Extrinsic motivation is involved when a reward is associated with performing the activity. In this case, the contributor expects to obtain something back from contributing to the community such as satisfying a software need, recognition, skill improvement, career advancement, or even being paid in some cases [Lakhani and Wolf, 2005].

The Boston Consulting Group [Lakhani et al., 2002] found that people contribute to F/OSS for different reasons. (Figure 3). The intellectual challenge and skill improvements are the main reasons why most OS community members contribute. Results of the study also show that contributors fall into four groups:

- The F/OSS believers, who contribute because they believe all source code should remain open.
- The thrill seekers, who contribute because of the intellectual stimulation that is associated with programming and view it as a hobby.
- The skill enhancers, who contribute mainly to improve their programming skills.
- The IT professionals, who contribute because of work related needs and for professional status improvement.

Reasons for contribution can also be explained in economic terms. Total benefit for a contributor is the sum of immediate payoff (benefits and costs) and delayed payoff (benefits and costs). Immediate benefits include satisfaction from use of the software and cost comes from opportunity

cost of time spent on programming. Delayed payoffs are future career opportunities and ego gratification stemming from a desire for peer recognition [Lerner and Tirole, 2002].

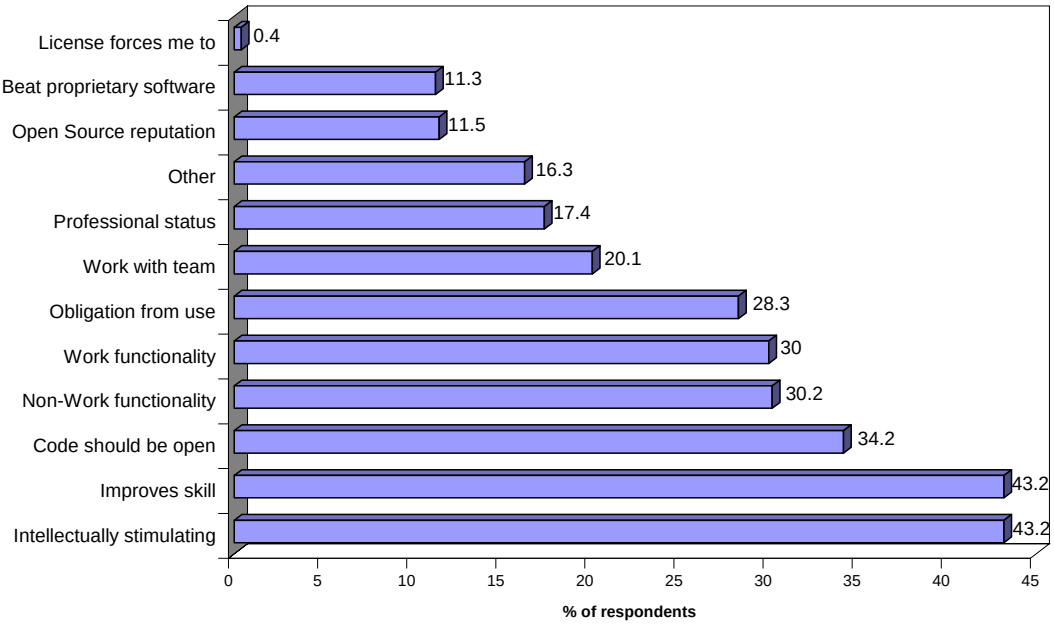

Figure 3: Motivation in F/OSS Communities - Source BCG 2002

Community Structure.

As shown in Figure 4, an F/OSS community has an onion-like structure that is based on the level of contribution [Cox, 1998, Gacek et al., 2001, Mockus et al., 2002, Moon and Sproull, 2000]. The core is the smallest group in the community that is responsible for the majority of code development (about 80% of source code is generated by the core) and effort contribution [Crowston and Howison, 2005, Krishnamurthy, 2002, Mockus et al., 2002]. It also maintains the most control on what features should be in the product and how it should be designed [Lee and Cole, 2003]. Co-developers surround the core. They contribute occasionally by modifying or reviewing code or submitting bug fixes. Surrounding the core and the co-developers are the users who are the majority in the community. Users can be active or passive users. Active users are the ones that use the latest releases and usually contribute ideas and bug reports. Passive users are free riders who simply use the software without contributing back to the community [Crowston et al., 2004].

Traditional software development is a team endeavor focusing on the development of large software systems through a software development life cycle that consists of a set of stages: System Planning, Analysis, Design, and Implementation. In contrast to the traditional world of software engineering, open software development communities do not seem to adopt or practice traditional software development processes [Scacchi, 2001]. In this subsection, we examine the development processes being used in practice.

Research on F/OSS development processes in different project communities [Crowston and Howison, 2005, German, 2003, Johnson, 2001, Lee and Cole, 2003, Scacchi, 2002] so far found no globally accepted framework that defines how F/OSS is or should be developed [Scacchi, 2002]. However, many F/OSS projects displayed a great degree of similarity. Projects begin with

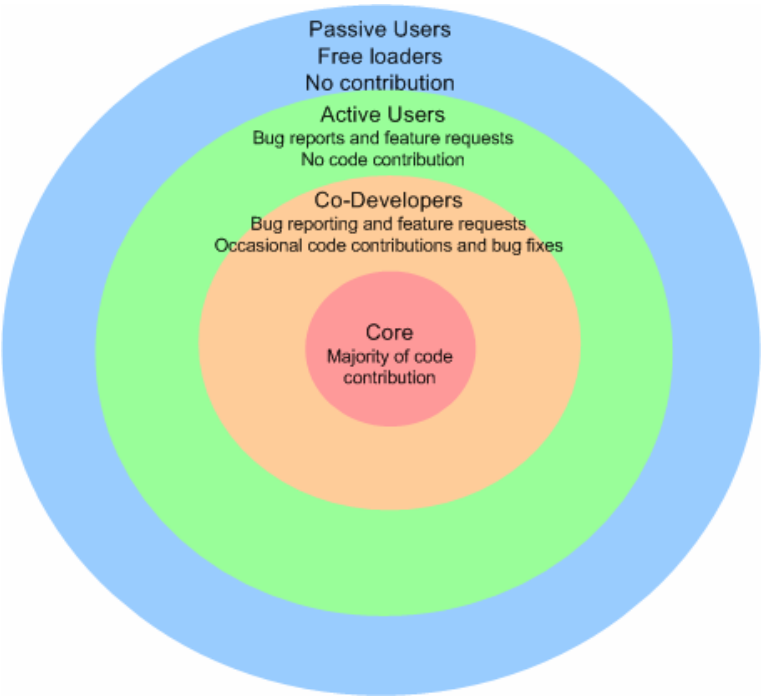

Figure 4. Community Structure [Crowston and Howison, 2005]

a prototype with pre-defined requirements developed from scratch or based on existent older product [German, 2003, Johnson, 2001, Scacchi, 2002]. Then, this early version incrementally evolves through rapid development iterations from the community, while concurrently managing as many designing, building, and testing activities as possible [Cockburn, 2002, Fowler, 2003, Johnson, 2001, Kogut and Metiu, 2001]. The five main steps for this approach are [Jorgensen, 2001]:

1. Code: potential F/OSS contributors take the initial step of submitting their code;
2. Review: talented and respected peers review the code in order to improve the quality of the code being submitted;
3. Pre-commit test: committers, the gatekeepers who are responsible for making code contributions permanent, test each contribution carefully;
4. Development release and parallel debugging: The committer incorporates the module in the development release if it passes the pre-commit test. The quality of F/OSS could be comparatively better than proprietary software, because the additional sets of eyeballs viewing the code might help catch additional bugs. Bugs can also be fixed by the individuals that identify them;
5. Production release: contributions eventually become part of the production release. They are merged into the stable production branch.

These development tasks can be classified into a two-tier structure that identifies several key tasks performed in the development process. These tiers are called Core and Periphery [Lee and Cole, 2003].

The core includes the tasks of selecting and retaining code for the official production release according to the pre-defined requirements. These tasks are executed by members with special privileges such as a project leader and maintainers.

The periphery includes the tasks of submitting source code and fixing bugs (performed by the Development Team) and reporting and documenting a bug (performed by the Bug Reporting

Team. The source code and fixed bugs are tested and reviewed by thousands of developers in the periphery. This process is referred to as peer review or gate-keeping in the F/OSS community. Peers evaluate code and provide suggestion for further improvement if it does not meet the requirements of the project.

As discussed in above, the open software development starts with a prototype and, in most cases, requirements of the software need to be changed through the requests of developers. For example, developers at the periphery can propose a new requirement because they want the feature and are willing to do most of the work. Then, the core decide on its value and chooses to accept or scrap the idea. [Hissam et al., 2001]. In a F/OSS development process, new requirements for software are described, asserted, or implied informally through an email message or within a discussion thread that is captured or posted on a project's Web site board for open review, elaboration, refutation, or refinement [Scacchi, 2002]. No matter what methods are used, new requirements are brought by the periphery, and are gathered and prioritized by the core [German, 2003]. Thus, on one hand, the core decides which new requirements are to be implemented and in which order; on the other hand, the periphery provides input and apply pressure on the core to shape their decisions. The two-tier task structure also coordinates activities of reformulation of the requirements for developing the F/OSS and ensures adaptation to shifting user requirements. These requirements are conveyed through informal communication mechanisms, such as online discussion forums or threaded email messages [Scacchi, 2001].

Note that the two-tier task structure is defined according to tasks not individuals. In reality, there exist no strict rule for a certain individual performing a specific task. For example, Lee and Cole [2003] report 49% of the Bug Reporting Team also performed the tasks of the Development Team, while 29% of the Development Team performed tasks of the bug reporting team. The overlap of the Development Team with the Bug Reporting Team shows that an individual can play different roles and perform multiple periphery tasks. This overlap is also true for individuals performing core tasks. In the majority of F/OSS projects, members that perform core tasks also contribute most of the code [Krishnamurthy, 2002]. However, the majority of F/OSS projects are not successful in the sense of creating a large enough community that is able to sustain itself. This failure might explain the tendency for development efforts to be centralized [Crowston and Howison, 2005]. Successful projects with larger development communities observe more distributed communication and development effort. However, it still remains true that the members performing core tasks are also responsible for generating more code [Crowston and Howison, 2005].

As shown in Figure 5, the two-tier task structure is combined with the five stages development process to show clearly how various tasks are completed in different stages. As noted, the solid line from stakeholder to stage means that specific stakeholder puts in effort and provides key input to a certain stage. The dashed line with a notion of transition reflects the reality that individuals can change their roles in the F/OSS development process by performing different tasks. Figure 6 shows a synthesis of the community structure with the task structure and illustrates how different member classes perform different tasks.

THE SOFTWARE

A continuous output of the unique development process is the resulting software. The software might have some unique benefits when compared to proprietary software. However, such benefits also depend on how potential users approach F/OSS to take advantage of it. Furthermore, each of the four components of F/OSS affect the quality of the software differently. F/OSS also affects how the users can benefit from the software (Section IV).

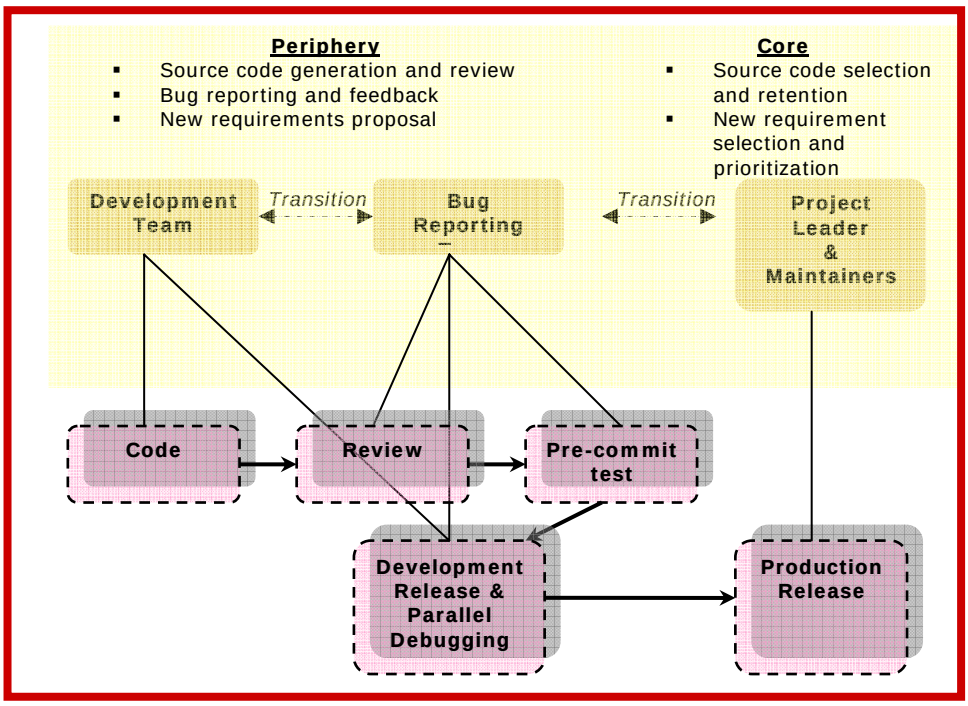

Figure 5: F/OSS Development Process and Two Tier Task Structure

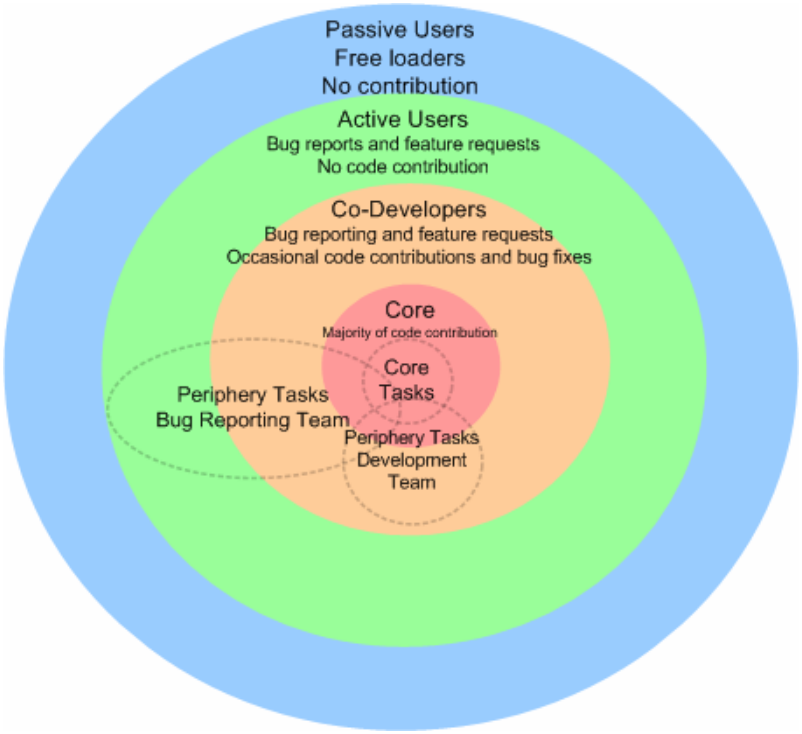

Figure 6. Community Structure Illustrating Who Performs Which Tasks

An important issue for F/OSS is modularity. A software module is a "portion of an application program that can be designed, developed, and tested relatively independently of the rest of the program" [Lee, 1999]. Software modularity promises a more flexible development process and shorter development cycles, and thus more robust products. For F/OSS, modular software design provides the foundation for the parallel development process by allowing independent development and testing of individual modules. Thus, modularity potentially shortens the development cycle and increases the robustness of the product. At the same time, however, such a flexible development process and the diverse composition of the community create challenges for F/OSS final products. Such problems are epitomized by Conway's Law, which states that "the structure of a product mirrors the structure of the organization that creates it." [Crowston et al., 2004]. Therefore, it might be harder to achieve an integrated product from the modules developed through different processes and physically distributed community members.

The IPO system conceptualization of the F/OSS is the key in understanding the eventual quality of the software output. As we saw earlier in this section, the observable development pattern for most F/OSS project is centralized [Crowston and Howison, 2005, Krishnamurthy, 2002]. It is different from the cathedral metaphor commonly used to describe the F/OSS development process [Raymond, 1999c]. Companies developing proprietary software might be able to replicate this centralized process for source code generation. However, the biggest difference is in the feedback. First order feedback, such as bug reports and the peer review process, ensures the software meets the requirements and quality standards of development. Second-order feedback is concerned with updating these requirements. In a proprietary development setting, the feedback is limited to the development team and the customer in some cases. Furthermore, in F/OSS if the community is large and active, the level of feedback will be difficult to replicate in a proprietary software development process economically.

In summary, the development process and community impact the quality of the software directly. A large community will be worthless if the members are not motivated to contribute source code and feedback to the development process. The license also impacts the type of contributors and quality of the feedback. This community feedback is what sets the F/OSS and proprietary software apart. Its greatest impact is on quality. Organization of the community and coordination of effort is tied to software modularity. As a result, the quality of the software also depends on the modularity of its design.

IV. BENEFITING FROM OPEN SOURCE

F/OSS is characterized by its four components[16] (Section II). Companies wishing to benefit from F/OSS can do so in different ways. We summarized these benefits in terms of three distinct approaches centered on each of three of the four components: software, community and license. These approaches lie on a rough continuum starting with no involvement in the community, and ending with the company as an F/OSS developer who initiates the community. In this Section we describe these approaches and key questions for each approach that companies should ask before investing. No matter which approach a company chooses to take, its success will depend on the quality of all four components, not only the one in focus.

SOFTWARE-CENTERED APPROACH

What is Software-Centered Approach and when should it Be Used?

The software-centered approach focuses on using the product, (i.e. the developed software) directly. The software to some extent can be viewed as a commodity. However, different from proprietary software, F/OSS, obtained free or purchased, is free for further revision and distribution. Software users can take advantage of the specific benefits of F/OSS as passive users in the community. Their role and degree of involvement in the community are minimal.

[16] For reference, the four components are license, community, development process, and software,

Their objective is mainly autonomous in-house software usage. They are not obligated to provide feedback or modifications back to the community.

As potential users, companies first clearly identify their needs in terms of objectives, budgets, time restrictions, and current technical capabilities (hardware, software, knowledge, and skills). They then evaluate the software candidates, based on their needs. This process is nothing new. Companies can use exactly the same procedures as for evaluating proprietary software. Their ultimate objective is to obtain software which is ready to be used or modified for use.

Potential users can obtain the software through three different approaches. First option is in the form of bundling. In that case, the desired software is bundled with hardware or as part of a software package. Users can purchase the software from hardware vendors, such as HP, which has "over 200 products that ship with open source software"[17]. A second option is to obtain the software from a F/OSS vendor such as Red Hat[18]. Users can download the software from Red Hat website for free. However, Red Hat charges for supporting services, through a subscription fee. For both the above approaches, the vendors guarantee support. Users do not need to reach the community directly but can obtain the software indirectly through the vendor. Alternatively, potential users can choose to download the software directly from the community. In this case, the users can obtain support from the community if needed. Such support is not guaranteed.

For all three approaches, two with indirect and the third with direct access to the community, it is important for users to be aware of the nature of the software, and the other components of the IPO model. For the first two approaches, even though the users do not connect with the community directly, the vendors who provide software to the users are members of the community. For any situation, the software is subject to the impact of the community. Users can harness the benefits of F/OSS in the future, such as frequent and free new release updating. For the third approach, the users themselves are members of the community. Therefore, in all cases, even for companies following the software-centered approach, understanding the license, development process, and community, in addition to the software product itself is important in making the adoption decision.

The evident benefit of the software-centered approach is reduction in time and effort. Companies can choose comparatively mature, or ready-to-use, F/OSS. This approach does not require involvement in the community and development process. The companies can be pure users, in other words, free-loaders. This approach does not require substantial technical capabilities from the companies either, since they need not be involved in the development process. However, less involvement and less input mean that they influence the software, the community, and the development process less. They are more like consumers and they must accept the restrictions set up by the licenses.

The software-centered approach is most appropriate for companies that need well-developed, highly commoditized software with a strict budget or timeline, or do not have the technical capability to contribute, or are not willing to get involved in the software development process.

Benefits

The software-centered approach entitles potential users to leverage any unique advantages of F/OSS, compared to proprietary software. Among the benefits of F/OSS, some most widely cited are reliability, security, and low cost.

Reliability. The F/OSS demonstrates a high level of reliability [Williams et al., 2005]. Compared to proprietary software, F/OSS development communities contain a virtually unlimited number of developers, not constrained as is the case with proprietary software vendors. Since all community

[17] http://opensource.hp.com/
[18] www.redhat.com

members can access to the source code and debugging tools, they can often suggest both bug fixes and enhancements to the source code and have their contributions reviewed by peers.

Security. Companies do not have to wait for vendors to release patches and security upgrades for their software because the community maintains the effort. In the same way the development process allows the production of more reliable software, it allows the production of more secure software. Identified security problems are communicated in the community and fixed in much shorter cycles than closed software. Some may argue that closed source is more secure because of its obscurity. However this assumption only gives a false sense of security and usually a longer period of time passes before the vendor is actually aware of the existence of a security problem. It might take even longer to fix [Raymond, 1999b]. Therefore, the wider and easier access to source code, which is considered insecure from the perspective of proprietary software development, actually increases the probability of detecting and fixing F/OSS errors and problems. This view is reflected in Linus' Law "Given enough eyeballs, all bugs are shallow." [Raymond, 1999c].

Low Cost. Advocates of F/OSS claim that it costs much less compared to proprietary software because licensing fees are eliminated. Administrative overhead is potentially reduced because accounting for copies in use is not needed. Cost of upgrading and maintenance is also reduced due to the free contribution of the development community and the improved stability and security of F/OSS. Based on numerous case studies, Stafford [Stafford, 2004] points out that F/OSS costs less than proprietary software especially in server environments. However, this analysis may not be the complete picture. Costs based on the total cost of ownership (TCO) including the total cost of acquiring, training, and customizing, might result in a different picture. To switch to F/OSS might not mean that TCO is reduced. Potential costs may involve a requirement of investment into some specialized expertise and training of employees to learn how to use the system, or even replacing the platforms that are in place. In some cases, these additional costs may not make it worthwhile to select F/OSS solutions [Stevenson, 2005].

Challenges

The unique development process and community composition of F/OSS creates challenges for potential users. Minoru Development SARL an F/OSS development, management, and consulting firm, suggests that the main challenges facing F/OSS projects are[19]:

1. Open source projects are not deadline driven, which may not be a problem when deploying finished work, but can be a problem if customers depend on anticipated future events. The best way for customers to manage this risk is to participate actively in the F/OSS project[20].
2. F/OSS is not as well established as proprietary software in some areas.
3. F/OSS involves a high entry barrier for non-technical users. The first success of F/OSS was in areas where the users and developers are one and the same. F/OSS is strongly technically orientated and is unproven for non-technical applications. Although open source is now expanding into new areas and producing products for non-technical users, this work is still in its infancy.

Support. For proprietary software, technical support is usually part of the purchased package. But for F/OSS, support experiences can vary. On one hand technical support might be prompt and helpful, with the whole community as the potential helper. On the other, desired support may not be available, since no one is obligated to provide help. This uncertainty implies that potential users of F/OSS evaluate the community before adopting the software. If this evaluation still posits a challenge, the potential users should consider the vendor-supported options.

---

[19] http://www.openhealth.com/en/opensource.html

[20] This recommendation aligns with the next suggested approach, community-centered.

All four components of F/OSS can be important success factors for its adoption. The community behind the software directly determines the quality of the software and the availability of support. The quality of the software itself depends on the development process. Since criteria for a good development process are not established , we expect that a healthy community can serve as proxy for a good development process, assessed by its size, openness, and diversity. Meanwhile, the license of the software restricts the application of the software. Therefore, before making the final decisions, potential users should evaluate not only the software, but also the corresponding community and license, to gain maximum benefits from F/OSS.

Table 1 provides a question-list for potential users, to help make decisions about software adoption when using the software-centered approach.

THE COMMUNITY-CENTERED APPROACH

What is the Community-Centered Approach and When Should it be Used?

A promising approach for companies is to benefit from the communities. The focus here is not only on adopting the final product, but also taking advantage of the community members who produce the product. A healthy community provides a much broader and richer knowledge database and more flexible development process [Wheeler, 2005]. The objective of this approach is to use the whole community as an external knowledgebase. Companies do not simply choose a software product and use it. Instead, they choose an appropriate community and become actively involved in the development processes and in community activities. Potential users can be co-developers or active users. They are acting as significant stakeholders in the community. They, themselves, are part of the F/OSS knowledge database. They originate opinions and comments, share codes, provide feedback, and give suggestions.

This approach is to some extent a hybrid of in-house development and outsourcing. Companies taking this approach need the technical capability to become involved in the software development process and contribute to the community. At the same time, their capability is not sufficient for them to develop the software all by themselves. This approach works better for companies that want to build their software development knowledge and skills, but need external intelligence to do so. These companies should aim to use and improve the software continuously, rather than the autonomous in-house use in software-centered approach.

Benefits

Like the software-centered approach, the community-centered approach can enjoy the benefits unique to F/OSS. Besides the software advantages, the community-centered approach further provides some level of influence upon the software development and features. As a result of active participation, involvement, and input, these active firms or co-developers are able to gain some influence on software development issues, such as key features of the software. Thus, they are able to use the community to customize the software to meet their needs.

At the same time, by participating in the development process, companies can improve their learning ability. Training is also embedded in the participation process [Lussier, 2004]. Companies can increase their understanding of the software development process, the strength and weakness of the software, and the feedback and reports from other users. Such valuable learning experience should enhance the companies' technical capability and facilitate the in-house application or customization of the software.

Challenges

This approach sets a higher bar for companies. To gain more control and influence, and to learn more, companies need to devote more time and effort. Companies should also posses certain technical capabilities to be able to act as active participant or co-developer in the community. They, as participants but not initiators, still must follow the restrictions of the licenses.

Table 1. Questions for Software Centered Approach

Category	Question	Desirable Answer
Community	➢ Size: what is the size of the community? Is it growing?	➢ Big or Growing. The larger the size of the community, the more feedback there will be. A higher quality product will be associated with a larger community, while the quality of the product will improve in the long run as the community shows signs of growth.
	➢ Dynamism: are both developers and users actively participating in the F/OSS development?	➢ Significant portions of the community are active participants. Active participation not only by developers but also by the users is what makes the difference in quality of the software. It is also the means by which community members provide technical support.
	➢ Knowledge: does the community (members) possess broad knowledge and expertise?	➢ The community is effective in using the knowledge its members possess. The community is effective in solving problems that arise. The development process is planned and structured. Problems with the software are easily communicated and solved in a short time cycle.
	➢ Users: are the users of the F/OSS from diverse industries/domains?	➢ The software has a broad user base. Users from different background might bring new perspectives and insights into F/OSS project. This broad user base is also reflective of the success of the software.
	➢ Core players: who are the core players/leaders of the community?	➢ The objectives of core player—Individual or commercial entities, are not contrary to that of organization. The organization will be able to deal or influence these individuals or entities should the need arises in the future.
Development Process	➢ Documentation: Does the community maintain a complete documentation of the F/OSS development? Does it keep record of all releases? Does it provide detail documents of notes and changes?	➢ Available documentation is not only a general guide but also a detailed manual that you could hand to a novice provided it is up to date. If not, then there needs to be a supportive community that can augment the need for documentation.
	➢ Bugs database: does the community maintain a bug database that provides a user-friendly interface for users to report bugs, and to search bugs and corresponding solutions provided by the developers?	➢ There is an available database that should be able to keep a record of all reported bugs, whether the bug has been fixed or not, which version of the software does the bug belong to, and whether the bug submitter has agreed that the bug has been fixed. This bug database is not only a good indicator for the health of the community, but also providing useful information for future revision and usage of the software.
License	➢ Can we accept the license fee, if any?	➢ Some F/OSS requires a license fee for use of the software in a commercial context. However, the license fee is just a small portion of TCO. The relatively low license fees of F/OSS need not necessarily reduce TCO of using and maintaining the system.
	➢ Can we accept the restrictions set up by the license?	➢ Licensing terms should be consistent with expectations, goals and risk tolerances of company. F/OSS licenses should be carefully reviewed before use of the software in a commercial context.
Software	➢ Do we have sufficient technical capabilities (hardware, systems, knowledge, and skills) to use or customize the software?	➢ If companies do not possess the required technical capabilities then they should be acquire these capabilities. As part of TCO evaluation, the costs required to obtain the necessary technical capabilities, such as hardware configuration, hiring new IT personnel should all be factored in. These capabilities can be searched for within the community.

Even with all the input, the desired level of control might not be guaranteed. In spite of the effort and time devoted, companies risk having little or no influence in the process. Therefore, companies must fully evaluate the community. As the primary condition for this approach, the community must be well organized, of high quality, and be friendly to newcomers.

Another challenge for community-centered approach is that in some cases companies need to adjust their own routines and structures to fit into the F/OSS development process. Such adaptations could lead to higher development costs and conflicts within the companies.

As with other approaches, all components of F/OSS should be balanced to maximize benefits gained from this community-centered approach. The companies should try to estimate the degree of influence they require and might obtain if they participate actively and contribute to community activities. This effect can be gauged based on information from previous members and the structure and mechanism of the community. They should assess the license problem, in terms of such factors as source code disclosure and authority to modify and/or resell.

Table 2 provides a question-list for potential users, to help make decisions about software adoption when using the community participation approach.

LICENSE-CENTERED APPROACH

What is the License-Centered Approach and When Should it be Used?

A less obvious approach that is suited for companies developing software is the license-centered approach. This approach involves initiating an F/OSS project by either releasing the software of an existing solution to the community as F/OSS, or initiating an F/OSS community to develop the software. The released source code will be the basis for future development of the software. The releasing company can act as an incubator for the project to see whether it will develop into a self-sustaining system. We call this approach the license approach because the company controls how it is going to license the software. Companies taking this approach will be acting as the core of this community and will be benefiting from the F/OSS development system as described in Section III.

As compared to the previous two approaches, this approach is concerned with developing a software product. It provides the company with the most control over the software development. The other two approaches are concerned with obtaining a software solution.

The two basic rationales behind using this approach are[West, 2003]:

1. It can provide the company releasing[21] its internal software with a level of developmental assistance that it can never afford to obtain or match on its own.
2. It can potentially improve the diffusion of the product and create network effects associated with this diffusion from which the company could benefit [West, 2003].

This approach has been employed by many organizations (such as Apple with the Darwin project[22], and SUN with the Open Office project [Gedda, 2005]) that released software or parts of it as F/OSS with varying degrees of success [Dahlander and Magnusson, 2005, Gedda, 2005, West, 2003]. By acting as the core of an F/OSS community, the company can tighten the relationship with customers, leverage customers as a resource, and improve the customization of the software for its customers [Hipple and Katz, 2002]. This approach can also help with building switching cost for users of the software. Therefore, it is most appropriate for companies that aim to improve their competitive position through the means of community building, or for companies that simply need external intelligence to develop and customize the software and want maximum control over the software development.

[21] We use the term releasing company in what follows.
[22] http://developer.apple.com/darwin/

Table 2. Questions for Community Centered Approach

Category	Question	Desirable Answer
License	➢ Can we accept the license fee, if any? ➢ Can we accept the restrictions set up by the license?	➢ Some F/OSS requires a license fee for use of the software in a commercial context. However, the license fee is just a small portion of TCO. The relative low license fees of F/OSS need not necessarily reduce TCO of using and maintaining the system. ➢ Licensing terms should be consistent with expectations, goals and risk tolerances of company. F/OSS licenses should be carefully reviewed before use of the software in a commercial context.
Community	➢ Size: what is the size of the community? Is it growing? ➢ Dynamism: are both developers and users actively participating in the F/OSS development? ➢ Knowledge: does the community (members) possess broad knowledge and expertise? ➢ Users: are the users of the F/OSS from diverse industries/domains? ➢ Core players: who are the core players/leaders of the community? ➢ Does the community have a clear structure and rules of organization? ➢ Does the community promote new participation? Are there any contribution guidelines or documentation to make it easier for newcomers to contribute? ➢ Do community members take the time to review the work of others?	➢ Big or Growing. The larger the size of the community, the more feedback there will be. A higher quality product will be associated with a larger community, while the quality of the product will improve in the long run as the community shows signs of growth. ➢ Significant portions of the community are active participants. Active participation not only by developers but also by the users is what makes the difference in quality of the software. It is also the means by which community members provide technical support. ➢ The community is effective in using the knowledge its members possess. The community is effective in solving problems that arise. The development process is planned and structured. Problems with the software are easily communicated and solved in a short time cycle. ➢ The software has a broad user base. Users from different background might bring new perspectives and insights into F/OSS project. This broad user base is also reflective of the success of the software. ➢ The objectives of core player—Individual or commercial entities, are not contrary to that of organization. The organization will be able to deal or influence these individuals or entities should the need arises in the future. ➢ The community is well organized into independent development teams focusing on different modules. There is also a communication structure that allows for the integration between the different teams. ➢ There are clear guidelines on how individuals should contribute. The source code is well documented. There are even technical documents that can help contributors get started. ➢ Peer review process is very active with short review cycles.
Development Process	➢ Do we have sufficient technical capabilities (hardware, systems, knowledge, and skills) to participate in the development process? ➢ Do we have to change our organizational structure or routines to participate?	➢ We have enough capabilities to contribute to the development effort. The idea to learn from this effort or increase our benefit from the final product. ➢ Our current organizational structure promotes the participation of our employees in F/OSS projects. Our employees are acquainted with F/OSS development methodologies If not, then change management is to be introduced to instill new values and introduce our employees to the F/OSS methodologies.
Software	➢ Is the software design modular?	➢ Software has a good modular design that enables a decentralized development process and allows for further expansion and growth of the development community.

released software or parts of it as F/OSS with varying degrees of success [Dahlander and Magnusson, 2005, Gedda, 2005, West, 2003]. By acting as the core of an F/OSS community, the company can tighten the relationship with customers, leverage customers as a resource, and improve the customization of the software for its customers [Hipple and Katz, 2002]. This approach can also help with building switching cost for users of the software. Therefore, it is most appropriate for companies that aim to improve their competitive position through the means of community building, or for companies that simply need external intelligence to develop and customize the software and want maximum control over the software development.

Benefits

In contrast to the other approaches, the license-centered approach is concerned with producing a product. This product will enjoy the same benefits and qualities that were discussed in the other approaches, provided the company is able to grow the F/OSS system into a healthy one. Unlike the other approaches however, the benefits that the companies can achieve from this approach are all long term. It takes a long time to grow the F/OSS system into a self-sustaining state where the F/OSS benefits can be realized in the product.

Provided the community reaches the critical self-sustaining size, the company can gain numerous benefits from the F/OSS development system [Bonaccorsi and Rossi, 2003b, Scacchi, 2004, West, 2003, West and Gallagher, 2004b]. It would help reduce the costs of development since community members from outside the company contribute to the development process of the product. These contributions include not only source code, but also new and innovative ideas. With the source code open, the software acts as a toolkit that customers can use to bring about new design innovations and improve customization. It helps the producer in better understanding customer needs, increasing customer satisfaction, and prolonging the product life cycle [Hipple and Katz, 2002] [Scacchi, 2004].

Furthermore, releasing software as F/OSS would speed up the diffusion of the product since there is virtually no cost to obtaining F/OSS. This approach can also place the releasing company in a better competitive position especially when the tipping of the network effect occurred in favor of the competition. Releasing a low cost alternative puts pressure on the competition to lower their prices. Furthermore, by supporting the establishment of a self-sustaining system to overlook the development and improvement of the software, the product could be continuously improved to replicate any features that are already included in the competitors' product, or even surpass them. A self-sustaining system would also mean that the pressure is on the competition to innovate continuously, with minimum resource allocation on the part of the releasing company. The company can relocate previously allocated resources for the project to another, more profitable one. The opposite may not be true for a competitor using a proprietary development method, because the resources available within an F/OSS community can not be replicated economically in a proprietary setting. While the competitor might use the ideas to improve the code, it seems likely that imitation, will always lag the F/OSS community due to the embedded knowledge in F/OSS and the difficulty in producing at the same rate or quality as an F/OSS community.

This approach provides a good means for customer relationship management. By improving the quality of the software, the customized software can attract more customers. These customers or users of the software become part of the community. With network effects, better software leads to a bigger community, and a bigger community makes the software even better. This cycle pushes the adoption rate of the software. Meanwhile, by involving the customers in the development process, the companies are able to tighten the relationship with customers. This involvement of the customer also creates switching costs.

Challenges

As with other approaches, it is a matter of balancing the four components of F/OSS to achieve success using this approach.

The biggest issue with having a company release software as F/OSS is that it is relinquishing its right to appropriate any income directly from that software. If the software was a source of competitive advantage for the company then it does not make sense to create a spin-off community from it because the company would be losing a source of income. The companies need to determine what to release.

One strategy companies could use to deal with this matter is "open parts" which deals with distributing parts of the software. [West, 2003]. This strategy relates to the number of technologies or layers released from the software. Companies can choose to release commoditized layers that are not a source of competitive advantage and retain full control of the layers that can be a source of competitive advantage. This strategy will depend, however, on how modular and loosely coupled the software design is. A good example for this strategy is the Apple Darwin project[23].

Another strategy companies can use is to "partly open" the source code [West, 2003]. For this case, they need to make appropriate use of licenses. Companies can choose to disclose technologies that provide competitive advantage but put legal restrictions that can provide value for the customer but prevent competitors from using the technology. The Microsoft shared source initiative and SUN Java technology are examples for such strategies but are not pure F/OSS.

Some of the successful F/OSS projects use a dual licensing strategy. This strategy allows the company to release the software as both F/OSS and as commercial product when the F/OSS license is not appropriate for the customer. MySQL used such a dual licensing strategy. In 2005, MySQL served around four million F/OSS customers and around four thousand commercial ones[24]. Other companies, such as Red Hat, choose to keep the software as F/OSS but build a business model around selling complementary products or providing service for their products.

The biggest challenge that remains after releasing the software is getting the community to participate. This matter is not trivial as discovered by the OpenOffice project, which is faced with delays because it does not involve enough developers [Gedda, 2005]. As discussed in Section III, F/OSS developers would like to work on an interesting project but not something that will lead to a dead-end or with marginal impact [Lerner and Tirole, 2002a]. Firms need to balance between the protection of competitive cutting edge technology and the interests of the developers. However, with that said, companies may be the appropriate entities for incubating F/OSS projects because of their stake in the success of the technology and their initial implementation of the software to be released to the public. If the releasing company acts as the core, it helps the community pass two major hurdles: proof of concept implementation, and leadership. Furthermore, companies may be in a better position to offer incentives or provide career paths for F/OSS developers. This capability could make spin-offs more attractive for developers, especially when the technology itself is not that interesting to them [Dahlander and Magnusson, 2005, West and Gallagher, 2004a]. Therefore, spin off projects might have a higher probability to succeed and pass the initial stages of an F/OSS project when compared to community initiated F/OSS projects. However, for the project to develop into a self sustaining one requires foresight from the company that initiated the effort in setting up the control structures of the community and a willingness to hand off that control to the community [West and Gallagher, 2004a, West and O'Mahony, 2005].

Even if the project is interesting and getting a community behind it is possible, an additional challenge is coordinating the development process, especially when the company retains its own development process. For the software to have any chance of success, the releasing company should employ an F/OSS development process that enables community contributions. Change management is required to instill F/OSS development methodologies into the company.

---

[23] http://developer.apple.com/darwin/
[24] http://www.mysql.com/company/legal/licensing/faq.html

Due to the complexity of this approach, large companies which plan to start up a F/OSS community can use lawyers to help them choose the best license to fit their needs and objectives.

Table 3 provides a question-list for potential users, to help make decisions when initiating a F/OSS community.

In summary, potential users of F/OSS can benefit by following different approaches to F/OSS, each emphasizing a different component of the F/OSS, i.e. software, community, and licenses. These approaches present different challenges. Selection of an approach should be done carefully with full consideration of the companies' objectives and resource availability.

Table 3. Questions for License Centered Approach

Category	**Question**	**Desirable Answer**
Software	➢ Are we clear on why we want to release our software as F/OSS? ➢ Will the release of the source code impact our competitive advantage? Is our software design modular? Can it be broken into separate products? ➢ Is the software design modular?	➢ We would like to improve our competitive position by increasing the diffusion of our product and/or gain assistance in the development process. ➢ The released software or the parts that are to be released are commoditized. If not, then we are trying to stimulate the interest of the community and rethink our business model. ➢ Software has a good modular design that enables a decentralized development process and allows for further expansion and growth of the development community.
Community	➢ Are we able to motive people to participate in the community? ➢ Can we afford the time and effort to initiate the community and to participate?	➢ The software is cutting edge technology to stimulate the interest of programmers. Or the success of our customers is aligned with our own success, so we will see more active participation from our customers. If none of these is the case, then we can also create incentives such as paying or employing some of the active community members. ➢ The values and ideals of F/OSS are part of our culture. We understand F/OSS development methodologies and promote the participation of the community. We also understand that developing an F/OSS community is a long term endeavor.
License	➢ Can we establish appropriate license to ensure our benefit? What implications will our choice have on community participation?	➢ We are taking the GPL approach to attract the free software believers. We can chose to use a dual licensing strategy so we can meet the demands of our commercial customers. We can also take the Mozilla license approach and attract members who seek their own commercial success to drive their participation.
Development Process	➢ Do we have sufficient technical capabilities (hardware, systems, knowledge, and skills) to participate in the development process? ➢ Do we have to change our organizational structure or routines to participate?	➢ We have enough capabilities to contribute to the development effort. The idea is to learn from this effort or increase our benefit from the final product. ➢ Our current organizational structure promotes the participation of our employees in F/OSS projects. Our employees are acquainted with F/OSS development methodologies. If not, then change management is to be introduced to instill new values and introduce our employees to the F/OSS methodologies.

V. MISCONCEPTIONS ABOUT F/OSS

In this section, we revisit many aspects of our discussion of F/OSS by examining some commonly held misconceptions. We first state the misconception and then briefly draw from our prior discussion in refuting them.

MISCONCEPTION 1: F/OSS IS LINUX.

One of the biggest misconceptions about F/OSS is that F/OSS is all about Linux [Moreira, 2002] [Reijswoud and Topi, 2003]. Linux certainly played an important role in the origins and development of the F/OSS. As one of the most popular server platforms over the past 10 years, it is perhaps the most widely used F/OSS. However, F/OSS is not just about Linux. It is more than just software as can be seen in Section III where we conceptualized the F/OSS IPO System. F/OSS is contributing to the development of different types of software. For example, more than 100,000 F/OSS are available for download at SourceForge.net alone[25]. F/OSS ranges from server software (such as network operating systems, database systems, and email and web severs), to desktop

applications (such as email clients, web browsers, and spreadsheets), to web applications (such as discussion forum, online surveys, and online content management systems). F/OSS is also not limited to Linux-based software. Many F/OSS also exist for Windows and other platforms.

MISCONCEPTION 2: F/OSS IS FREE.

F/OSS is free, but not necessarily totally free, depending on how you interpret free. It is a common misconception that F/OSS is free in the sense that no costs are involved [Viega, 2002] [Fitzgerald and Kenny, 2003]. As discussed in Section III, the free concept in F/OSS refers more specifically to freedom: free as in free speech, rather than free beer. F/OSS gives everyone the freedom to access the source code and redistribute copies. F/OSS also allows organizations and individuals to control how they use and adapt the source code to suit their business or personal needs. In the sense of free beer, although most of F/OSS can be redistributed for no cost, according to the FSF and OSI, nothing prevents a software producer from demanding a fee for distributing a copy of their software. Some producers may provide licensing for commercial/business use. Initial acquisition cost varies, from free to some amount. However, such costs only consist of a small percentage of the TCO, which includes all the costs related to deploying software, such as cost of installation, hardware configuration, technical support and IT personnel. It is TCO that should be taken into consideration when evaluating F/OSS costs.

MISCONCEPTION 3: F/OSS IS JUST THE SOFTWARE.

The first thing that might come to many people's mind when F/OSS is mentioned is software. They describe F/OSS with statements like, "It's no big deal; it's just software." [Pavlicek, 2003] However, F/OSS is more than just source code. We need to see the rich and valuable community, knowledge resource, and the development process and methodologies behind it, which, if used correctly, can improve product development and innovation. By taking these factors into consideration, the organization's role in the creation and use of software is vastly expanded. As discussed in Section IV, through a community- centered approach or a licensing-centered approach, organizations can increase their levels of participation and, therefore, maximize the software's relative fitness [Maher, 2000].

[25] http://www.ostg.com/pdfs/SourceForgeProjects_PR_Final1.pdf

MISCONCEPTION 4: ALL F/OSS ARE CREATED EQUAL

Literature advocating F/OSS often implies that F/OSS approaches would always yield higher quality products [Williams et al., 2005] [Raymond, 1999a]. As we discussed in Section IV, F/OSS projects can be of higher quality than their proprietary alternatives because of their development methodology. Furthermore, for software to benefit from F/OSS development methodologies, a motivated community should be working on the project and the efforts of the participants should be well organized and coordinated. However, studies shown that not all F/OSS projects inherit these practices and characteristics [Crowston and Howison, 2005], and no standard development process has been established [Scacchi, 2002]. F/OSS projects with different (or even the same) levels of participation, communication, organization, and control may differ in quality. Therefore, it will not always be the case that F/OSS projects provide higher quality. Therefore, we advocate that whatever approach an organization chooses to invest in F/OSS, it needs to evaluate the characteristics (such as size, knowledge, users, core players, dynamism, and openness) of the community.

MISCONCEPTION 5: NO SUPPORT OR TRAINING IS AVAILABLE FOR F/OSS.

Technical support is the primary concern of F/OSS users [Wheatley, 2004]. Without a specific vendor, users worry about who they can ask for help when things go wrong. As we discussed, many successful F/OSS projects involve a large community of developers, Internet mailing list, and archives. The breadth of resources available makes F/OSS technical support prompt and helpful. Many companies, such as Red Hat, built their business models on the support and training of F/OSS solutions. However, support and training also depends on the size and culture of the community. These factors, again, suggest that organizations should evaluate the community before adopting the F/OSS.

VI. CONCLUSION

F/OSS provides the freedom for anyone to acquire source code, inspect, verify, modify, use, and create derived works. However, it is more than just source code. It is a powerful community of talented individuals, a development process with different tasks and roles, and a set of software licenses. When evaluating F/OSS solutions, all four of these components need to be considered.

F/OSS is changing the way companies develop, acquire, and manage software at every level, from operating systems to applications. Business managers should evaluate open-source alternatives to proprietary software to explore the potential opportunities F/OSS may bring to business. Managers should also understand how to benefit from F/OSS and realize the challenges to determine which approach is more suited for their company's needs.

It is our hope that this tutorial will become a useful resource for managers to better understand the open-source movement, its components, its benefits, and the challenges of the different F/OSS approaches. We do not believe that this movement will abate. Its metamorphosis over the coming years will be fascinating.

ACKNOWLEDGEMENTS

We gratefully acknowledge Aaron Malcom and the anonymous reviewer for providing valuable insights and helpful suggestions.

Editor's Note: This article was received on July 16, 2005. It was with the authors for two revisions. It was published on November __, 2005.

REFERENCES

EDITOR'S NOTE: The following reference list contains the address of World Wide Web pages. Readers who have the ability to access the Web directly from their computer or are reading the paper on the Web, can gain direct access to these references. Readers are warned, however, that

1. these links existed as of the date of publication but are not guaranteed to be working thereafter.

2. the contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced.

3. the authors of the Web pages, not CAIS, are responsible for the accuracy of their content.

4. the authors of this article, not CAIS, are responsible for the accuracy of the URL and version information.

5. Date of last accessed is shown in parentheses at the end of the reference

Bezroukov, N. (1999) "Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)," *First Monday* (4) 10.

Bonaccorsi, A. and C. Rossi (2003a) Licensing Schemes in the Production and Distribution of Open Source Software. An Empirical Investigation, in *Free/Open Source Research Community*.

Bonaccorsi, A. and C. Rossi (2003b) "Why Open Source Software Can Succeed," *Research Policy* (32) 7, pp. 1243-1258.

Cockburn, A. (2002) *Agile Software Development*. Boston, MA: Addison-Wesley.

Cox, A. (1998) "Cathedrals, Bazaars and the Town Council," Slashdot, http://slashdot.org/features/98/10/13/1423253.shtml (June 19th, 2005).

Crowston, K., H. Annabi, and R. Heckman. (2004) A Structurational Model of the Dynamics of Free/Libre. *Presentation at the IFIP WG 8.2 OASISWorkshop, Washington, DC, 2004*.

Crowston, K. and J. Howison (2005) "The Social Structure of Free and Open Source Software Development," *First Monday* (10) 2.

Dahlander, L. and M. G. Magnusson (2005) "Relationships between Open Source Software Companies and Communities: Observations from Nordic Firms," *Research Policy* (34) 4, pp. 481-493.

Feller, J. and B. Fitzgerald (2002) *Understanding Open Source Software Development*. London, UK: Addison-Wesley.

Fitzgerald, B. and T. Kenny. (2003) Open Source Software in the Trenches: Lessons from a Large-Scale Oss Implementation. *24th International Conference on Information Systems, 2003*.

Fowler, M. (2003) "The New Methodology," http://martinfowler.com/articles/newMethodology.html (May 20th, 2005).

Gacek, C., T. Lawrie, and B. Arief (2001) "The Many Meanings of Open Source," Interdisciplinary Research Collaboration in Dependability, Technical Report 1, http://www.dirc.org.uk/publications/techreports/papers/1.pdf.

Gedda, R. (2005) "Lack of Developers Delays Openoffice.Org," *Computer World* April, 20.

German, D. M. (2003) "The Gnome Project: A Case Study of Open Source, Global Software Development," *Software Process Improvement and Practice* (8)pp. 201-205.

Glass, R. L. (2004) "A Look at the Economics of Open Source," *Communications of the ACM* (47) 2, pp. 25-27.

Grantham, T. (1999) "An Open Source of Business Opportunities," *Computer Dealer News* Feb 5.

Hein, G. (2004) "Open Source Software: Risks and Rewards," ECAR Symposium, http://www.educause.edu/ir/library/pdf/ECR0405.pdf.

Hertel, G., S. Niedner, and S. Herrmann (2003) "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel," *Research Policy* (32) 7, pp. 1159-1177.

Hickman, A. (2004) "How Do You Decide If Oss Works for You?" Nonprofit Open Source Initiative, http://www.nosi.net/node/27.

Hipple, E. V. and R. Katz (2002) "Shifting Innovation to Users Via Toolkits," *Management Science* (48) 7, pp. 821-833.

Hissam, S., C. B. Weinstock, D. Plakosh, and J. Asundi. (2001) *Perspectives on Open-Source Software*. Software Engineering Institute,Carnegie Mellon University, Technical Report CMU/SEI-2001-TR-019.

Johnson, K. (2001) A Descriptive Process Model for Open-Source Software Development, University of Calgary.

Jorgensen, N. (2001) "Putting It All in the Trunk, Incremental Software Development in the Freebsd Open Source Project," *Information Systems Journal* (11)pp. 321-336.

Kogut, B. and A. Metiu (2001) "Open Source Software Development and Distributed Innovation," *Oxford Review of Economic Policy* (17) 2, pp. 248-264.

Krishnamurthy, S. (2002) "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," *First Monday* (7) 6.

Lakhani, K. R. and B. Wolf (2005) Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects, in J. Feller, B. Fitzgerald, S. Hissam, and K. R. Lakhani (Eds.) *Perspectives on Free and Open Source Software*, Cambridge, MA: MIT Press.

Lakhani, K. R., B. Wolf, J. Bates, and C. DiBona (2002) "The Boston Consulting Group Hacker Survey," http://www.bcg.com/opensource/BCGHackerSurveyOSCON24July02v073.pdf.

Lee, G. K. and R. E. Cole (2003) "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development," *Organization Science* (14) 6.

Lee, S. H. (1999) "Open Source Software Licensing," http://cyber.law.harvard.edu/openlaw/gpl.pdf.

Lerner, J. and J. Tirole (2002) "Some Simple Economics of the Open Source.," *The Journal of Industrial Economics* (2) L, pp. 197-234.

Lerner, J. and J. Tirole (2002a) "Some Simple Economics of the Open Source.," *The Journal of Industrial Economics* (2) L, pp. 197-234.

Levesque, M. (2004) "Fundamental Issues with Open Source Software Development," *First Monday* (9) 4.

Lussier, S. (2004) "New Tricks: How Open Source Changed the Way My Team Works," *IEEE Software* (21) 1, pp. 68- 72.

Maher, M. (2000) "Open Source Software: The Success of an Alternative Intellectual Property Incentive Paradigm," *Fordham Intellectual Property, Media & Entertainment Law Journal* (2000) Spring.

McGowan, D. (2001) "Legal Implications of Open Source Software," *University of Illinois Law Review,* (2001) 1, pp. 241.

Mockus, A., R. Fielding, and J. Herbsleb (2002) "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology* (11) 3, pp. 309–346.

Moon, J. and L. Sproull (2000) "Essence of Distributed Work: The Case of the Linux Kernel," *First Monday* (5) 11.

Moreira, C. F. (2002) "Malaysia's Open Source Push," Aldrix News, http://www.aldrich.com.my/modules.php?name=News&file=print&sid=129.

O'Reilly, T. (1999) "Lessons from Open-Source Software Development," *Communications of the ACM* (42) 4, pp. 33-37.

Orr, K. (1998) "Data Quality and Systems Theory," *Communications of the ACM* (41) 2, pp.66-71.

Pavlicek, R. (2003) Open Source Perspective: The Significance of Open Source, in *Processor*, vol. 25, pp. 7.

Raymond, E. S. (1999a) A Brief History of Hackerdom, in 1st edition C. DiBona, S. Ockman, and M. Stone (Eds.) *Open Sources*, Cambridge, MA: O'Reilly.
Raymond, E. S. (1999b) "The Case of the Quake Cheats," http://www.catb.org/~esr/writings/quake-cheats.html (June 30th, 2005).
Raymond, E. S. (1999c) The Cathedral & the Bazaar Musings on Linux and Open Source by an Accidental Revolutionary, 1st ed. edition. Cambridge, MA: O'Reilly.
Raymond, E. S. (2000) "Revenge of the Hackers," http://www.catb.org/~esr/writings/cathedral-bazaar/hacker-revenge/ (Sep 1st, 2005).
Reijswoud, V. and C. Topi (2003) "Alternative Routes in the Digital World: Open Source Software in Africa," Free/Open Source Research Community Online Paper, http://opensource.mit.edu/papers/reijswoudtopi.pdf.
Scacchi, W. (2001) Software Development Practices in Open Software Development Communities. *1st Workshop on Open Source Software Engineering, Toronto, Ontario., 2001*.
Scacchi, W. (2002) "Understanding the Requirements for Developing Open Source Software Systems," *IEEE Software* (149) 1, pp. 24-39.
Scacchi, W. (2004) "Free and Open Source Development Practices in the Game Community," *IEEE Software* (21) 1, pp. 59- 66.
Stafford, J. (2004) "Evaluate Open Source, or Else," SearchEnterpriseLinux.com, http://searchenterpriselinux.techtarget.com/qna/0,289202,sid39_gci999458,00.
Stevenson, R. (2005) Study Shows Microsoft, Linux Costs Neck-and-Neck, in *Reuters*.
Surman, M. and J. Diceman (2004) "Choosing Open Source: A Guide for Civil Society Organizations," http://www.commons.ca/articles/fulltext.shtml?x=335#benefits.
Viega, J. a. B. F. (2002) "Dispelling Myths About the Gpl and Free Software," Cyberspace Policy Institute, http://www.cpi.seas.gwu.edu/oss/cpi_rebuttal.pdf.
Wayner, P. (2000) *Free for All*. New York: HarperCollins.
West, J. (2003) "How Open Is Open Enough? Melding Proprietary and Open Source Platform Strategies," *Research Policy* (32) 7, pp. 1259-1285.
West, J. and S. Gallagher (2004a) Key Challenges of Open Innovation:Lessons from Open Source Software.
West, J. and S. Gallagher (2004b) "Key Challenges of Open Innovation:Lessons from Open Source Software," http://www.cob.sjsu.edu/WEST_J/Papers/WestGallagher2004.pdf.
West, J. and S. O'Mahony. (2005) Contrasting Community Building in Sponsored and Community Founded Open Source Projects. *Annual Hawai'i Internationa Conference on System Sciences, Waikoloa, Hawaii, 2005*.
Wheatley, M. (2004) "The Myths of Open Source," *CIO Magazine* March 1st.
Wheeler, D. (2005), http://www.dwheeler.com/oss_fs_eval.html.
Williams, J., P. Clegg, and E. Dulaney (2005) The Advantages of Adopting Open Source Software, in *Expanding Choice: Moving to Linux and Open Source with Novell Open Enterprise Server*: Novell Press.
Wong, K. and P. Sayo. (2004) *Free/Open Source Software: A General Introduction*. United Nations Development Programme, Asia-Pacific Development.

APPENDIX I. RECOMMENDED READING RESOURCES

Free Software Foundation and Richard Stallman

- http://www.fsf.org
- Stallman, R. (2002) “Free Software, Free Society: Selected Essays of Richard Stallman”, Gay, J. (ed.), Boston, MA: Free Software Foundation GNU Press
- Williams, S. (2001) “Free as in Freedom: Richard M. Stallman’s Crusade for Free Software”, San Francisco,CA: O’Reilly Press. This is a biography of RMS.

Open Source Initiative and Eric Raymond

- http://www.opensource.org

- http://www.catb.org/~esr (includes a section from the Cathedral and Bazaar and various writings by Eric Raymond)

F/OSS Research

- http://opensource.mit.edu
- http://www.isr.uci.edu/research-open-source.html
- http://floss.syr.edu/
- http://www.firstmonday.org/
- http://opensource.ucc.ie/

Web resources for F/OSS

- http://www.dwheeler.com
- http://opensource.oreilly.com/

F/OSS news

- http://slashdot.org/
- http://www.newsforge.com/

APPENDIX II: A SAMPLE OF ESTABLISHED F/OSS PROJECTS

Project	Brief Description	URL
Redhat	A premier Linux and open source provider.	http://www.redhat.com/
SourceForge.net	F/OSS development websites. Contain various projects.	http://sourceforge.net/index.php
Freashmeat.net	F/OSS development websites. Contain various projects.	http://freshmeat.net/
OSDir.com	F/OSS development websites. Contain various projects.	http://osdir.com/
BerliOS.com	F/OSS development websites. Contain various projects.	http://berlios.com
Apache Software Foundation	Support various Apache projects	http://www.apache.org/
FireFox	Internet browser, which has more than 80 million users	http://www.mozilla.org/products/firefox/
OpenOffice	Similar to Windows Office. Compatible with all other major office suites.	http://www.openoffice.org/
Knoppix	A bootable Live system on CD or DVD, consisting of a representative collection of GNU/Linux software. It enables you to run Linux without installing anything on harddisk.	http://www.knopper.net/knoppix/index-en.html
Blender	The software for 3D modeling, animation, rendering, post-production, interactive creation and playback	http://blender3d.org/cms/Home.2.0.html
Php	A widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.	http://www.php.net/
Python	An object-oriented programming language…running on many brands of UNIX, on Windows, OS/2, Mac, Amiga, and many other platforms	http://python.org/
VideoLAN	Video streaming software	http://www.videolan.org/
Voip-info	VOIP applications software	http://www.voip-info.org/wiki-Open+Source+VOIP+Software

Compiere	ERP software with integrated CRM solutions	http://www.compiere.org/
GRASS GIS	A Geographic Information System (GIS) used for geospatial data management and analysis, image processing, graphics/maps production, spatial modeling, and visualization	http://grass.itc.it/
Claroline	An e-Learning application based on PHP/MySQL	http://www.claroline.net/
GnuCash	Accounting software, to track bank accounts, stocks, income and expenses.	http://www.gnucash.org/
Sakai	Similar to Blackboard, the software for developing a new Collaboration and Learning Environment (CLE) for higher education.	http://www.sakaiproject.org/
R	Statistical computing software, similar to S-Plus/SAS/SPSS.	http://www.r-project.org/

ABOUT THE AUTHORS

Mohammad AlMarzouq is a Ph.D. Candidate in the MIS track of the Department of Management at Clemson University. His research interests include knowledge management, project management, and software development with a particular interest in F/OSS.

Varun Grover is the William S. Lee (Duke Energy) Distinguished Professor of IS at the College of Business & Behavioral Sciences, Clemson University. He has published extensively in the IS field, with three books and over 150 publications in refereed journals. His current research focuses on the impact and effectiveness of IS at the organizational and market levels. Five recent articles have ranked him among the top five researchers based on publications in top IS journals over the past decade. His work has appeared in journals such as ISR, MISQ, JMIS, CACM, Decision Sciences, IEEE Transactions, California Management Review, among others. He is currently a Senior Editor for JAIS, MISQ (2006) and Database; Associate Editor for a JMIS, JOM, and IJEC and on the Board of Editors of numerous others.

Guang Rong is a Ph.D. candidate in the MIS track of the Department of Management at Clemson University. Her research interests include knowledge management, IT workforce, and IT valuation.

Communications of the Association for Information Systems

ISSN: 1529-3181