The Flow of Knowledge in Free and Open Source Communities

Daniel M. German
Software Engineering Group
Department of Computer Science
University of Victoria
Victoria, Canada
dmgerman@uvic.ca

Abstract

In this paper we present a survey of the methods used by a selection of successful free and open source projects to exchange, store and retrieve knowledge. In particular, we look into mailing lists, Internet Relay Chat, conferences, and code reviews. We explore how historical records left during the development become stored knowledge that can be subsequently retrieved. We also discuss the existence of meta-communities (composed of members of different communities) that allow knowledge to flow from one community to another.

1. Introduction

Free and open source software (FOSS) development has established itself as an effective way to develop software. Perhaps one of its most radical features is that its members are willing to give away knowledge without any direct remuneration[1]. This is particularly striking in an era in which intellectual property (mainly in the form of copyright and patents) is highly protected for its economic value. For this reason FOSS has been frequently compared to science where its participants publish their findings into a commons for the benefit of everybody [24].

The "community" is an important concept in FOSS development. It refers to the individuals and organizations that participate in the development of a FOSS application. Some participants are totally passive (by strictly using an application without ever participating in its development) to totally active (the so called "core" developers who are responsible for most of the contributions to the project. The ways in which a member of the community can participate in the development of a project are extremely wide. Table 1 lists various types of knowledge contributions that individuals can make. Organizations (beside paying developers to contribute to a FOSS projects) can contribute knowledge directly to FOSS projects. For example the original source code of Mozilla was donated by Netscape Corporation; and IBM has pledged to license 500 patents to open source projects [9]).[2].

One of the main challenges that FOSS projects have is the need to attract and nurture new members. It is, therefore, important to reduce the learning curve of newcomers to ease their integration into the development process and to encourage them to start contributing to the application as easily as possible.

In this paper we are interested in answering the following research questions:

- What are the methods used by a FOSS community to share, store, and diffuse knowledge?
- Is knowledge exchanged between communities, if so, how?

The methodology we have used is based on three main components:

- A literature review regarding how knowledge is created and diffused in FOSS communities.
- A qualitative analysis of the following successful FOSS projects: Apache, Evolution, Linux, GNOME, Mozilla, gcc and postgreSQL.

1 There are many individuals who are paid by a third party to contribute to a FOSS project. In this case we can consider these individuals as "proxies" of the organization who hire them. These organizations are, therefore, contributing knowledge without expecting a direct remuneration for their contributions–but there are most likely seeking indirect benefits.

2 Organizations can also indirectly contribute knowledge in the form of training and diffusion, and by paying employees to become contributors.

Type of contribution	**Description**
Source code	This is perhaps the most visible contribution.
Documentation	In the form of Web sites, user and developer manuals, magazine and Web articles, books, FAQs, etc.
Internationalization	Translations of the software and documentation into different languages.
Code Reviews	The discussion and improvement of source code contributions.
Testing and debugging	Formal or informal testing and debugging.
Bug reports	Submit bug reports that can be used by the development team to track and fix defects.
Configuration management and build process	Tasks required to maintain the environment necessary for multiple developers to participate.
Distribution of binaries	Preparation of binaries for download by any user interested to try the software.
Suggestions	Ideas on how to improve the product.
Answers to developer's questions	They help other developers who are contributing.
Answers to user's questions	They help individuals who are trying to use the software.
Release management	Dedicated to prepare and advertise new releases.
Legal	They provide information regarding legal issues, such as licensing, and other intellectual property issues.
Web site development and maintenance	These contributions usually gather knowledge from other sources and make sure it is persistent. It can also include those who contribute to wikis.
"Pointers" to knowledge	Perhaps the smallest type of contribution it involves answering a question by "pointing" to another source of information (such as a Web site or a research article).
Distribution packaging	Knowledge needed to prepare packages to be included in distributions (such as SUSE, Red Hat, Fedora, etc).

Table 1. Type of Knowledge Contributions to a FOSS project

- The experiences of the author as a contributor to several FOSS projects [3].

The paper is divided as follows. Section 2 addresses the question of how knowledge is shared, stored and diffused within a FOSS community. Section 4 analyzes how knowledge is exchange across FOSS communities. We conclude with a discussion of our findings and directions for future research.

2. How does knowledge flow within a FOSS community?

Research has shown that two important motivations that individuals have to become FOSS software developers are to: 1) improve their career perspectives (by acquiring and refining skills) and 2) be recognized in the meritocracy of a FOSS community[14, 8]. This implies the existence of knowledge flow in FOSS communities from those who have it to those who seek it. Given the variety of knowledge required to produce software (programming skills, application domain, management skills, marketing skills, etc) individuals might become both a producer and a consumer of knowledge depending on the skills that they bring to a given project (and the skills that they are particularly interested in learning and improving).

The flow of knowledge from one individual to another requires the creation and development of an infrastructure that supports it. It is also necessary to create mechanisms that permit its short and long term storage and retrieval.

From its beginning the Internet and FOSS have coexisted in a symbiotic manner: the Internet was born thanks to the sharing of source code and source code has thrived as the Internet matures. It is undeniable that the Internet is the main channel over which knowledge flows within a FOSS community. Project-sponsored conferences (see section 2.3) are perhaps the only form of exchange of knowledge in FOSS that does not require the Internet (although it uses it for its organization).

As FOSS projects evolve their communities evolve as well: new members join, some leave. At the same

3 The author is currently one of the core developers of Panotools. Panotools is a group of tools to combine two or more photographs into a panoramic one, see `panotools.sourceforge.net`.

time some of its members shift their roles, depending on many factors, including the time they can invest to the project [18]. Nakakoji et. al use *Legitimate Peripheral Participation* theory (LPP) to explain this evolution: "a community of professionals evolves by reproducing itself when peripheral new members (i.e. apprentices) become fully qualified members (i.e. masters). The process of becoming a master is the process of learning. [...] the community member acquires the skills and knowledge embodied in the community by interacting with master members" [18]. One important conclusion of the study by Nakakoji et. al is that the evolution of FOSS communities is determined by two factors: "the existence of motivated members who aspire to play roles with large influence, and the social mechanism of the community that encourages and enables such individual role changes" [18].

Like in any other software development team, members of a FOSS community eventually leave it for multiple reasons (these can range from lack of motivation or available time, to passing away). Without a constant influx of new members, any FOSS community will eventually collapse. A FOSS community, therefore, requires the flow of knowledge from one member to another, and the storage (temporary and permanent) of that knowledge so it can be retrieved and reused by new members (this knowledge becomes the project's community memory).

2.1. Email

Email is ubiquitous as a medium for the flow of knowledge in FOSS. A project usually starts with a mailing list that links developers and users (active–those that contribute to the discussions–and passive–those that only use the product without contributing anything in return). In an empirical examination of 100 FOSS projects Krishnamurthy found that most of them have very few contributors and, on the average, have 2 mailing lists [13]. As a FOSS community grows its discussions are split into different types of mailing lists.

The most commonly found lists are those dedicated to *announcements*, *users*, and *developers*. In the large projects that we analyzed we found that mature, widely used projects tend to have highly specialized mailing lists (Mozilla, for example, has 81). We discovered that there exist five main types of mailing lists:

- Announcements. Typically a low traffic, moderated list, it is intended to be used for announcements regarding the status and evolution of the project.
- Users support. Mailing lists dedicated to help members who have questions regarding how to use the product.
- Development. Developers use them to discuss the development of the project. Some project tend to have very specialized development lists (for example, Apache has a packagers- list for the discussion of issues related to how apache is packaged, distributed and made available to users).
- Software process related. The messages in these lists are usually produced by tools that support the development process (for example, a version control mailing list that has one message per source code commit, or a bug mailing list that has one message per bug reported).
- Documentation. These lists are dedicated to the discussion of documentation and the Web sites of a project.

We also found that all the surveyed projects archive their mailing lists and the majority provide some type of searching mechanism to them.

2.2. IRC

IRC (Internet Relay Chat) is an old Internet protocol that supports many-to-many instant communication. IRC has been widely used to link communities even before the advent of the World-Wide Web. Although it is rarely reported, many FOSS projects have IRC channels where different contributors can meet and exchange knowledge. Apache, for example, uses the IRC channel `#Apache` in `irc.freenode.net`[4]. The flow of knowledge in the Apache IRC channel is demonstrated by Rich Bowen (who is a member of the Apache Foundation and contributes documentation to the server). He writes a monthly column based on his experiences in the Apache's IRC channel[1].

GNOME has its own IRC server that hosts more than three dozen channels [5]. Similarly Mozilla maintains its own IRC server with more than two dozen channels (almost 60% of them are in languages different from English) [6].

IRC channels provide a very informal place to exchange information at all different levels. Perhaps its main drawback is that its discussions are rarely

4 `irc.freenode.net` holds several hundred IRC channels for FOSS projects, including postgreSQL, the Free Software Foundation, RedHat, and mySQL `http://freenode.net/primary_groups.shtml`.

5 `http://gnomesupport.org/wiki/index.php/IrcChannels`

6 `http://irc.mozilla.org/`

archived. They are similar to informal verbal conversations happening in the offices and halls of an organization.

2.3. Conferences

Several FOSS projects are organizing conferences where developers can meet face to face. Conferences are usually organized around presentations that are intended to exchange knowledge, or to train other contributors. They are also a place where discussions regarding the future of the project usually take place. From the projects that we surveyed these organize conferences:

- GUADEC. This is the GNOME Developers conference (it has taken place once every year since 2000)[7].
- ApacheCon. The Apache Conference, like GUADEC, has run every year since 2000. This year it has three versions: Europe, Asia and US[8]. GUADEC and ApacheCon are two of the oldest running conferences organized by a FOSS community.
- PostgreSQL Anniversary Summit. The 10 year anniversary of PostgreSQL is being marked with the project's first conference.[9]

2.4. Code reviews

Code reviews or code inspections were introduced by Fagan as a formal process in which the development team invests time and energy to review the code being produced [3]. In the most formal approach, code reviews are conducted during meetings for which the developers are expected to prepare. These meetings can result in the detection of defects or in recommendations on how the source code can be improved.

Because FOSS developers are usually geographically dispersed they are unable to conduct formal code reviews. Instead they conduct asynchronous reviews using email as the main communication channel. In an empirical study of software inspections Johnson and Tjahjono found no significant differences between meeting-based, and asynchronous code reviews. They did, however, found that the total effort required in meeting-based reviews was significantly higher when compared to asynchronous reviews (and hence the effort to find a bug was higher in meeting-based reviews) [10]. In FOSS code reviews have two main objectives:

- They minimize defects and provide better, cleaner code with less total effort.
- They improve the skills and knowledge of the reviewers and authors of the code.

Few FOSS projects have a formal process for code reviews, and those that do are usually mature, and expected to be reliable. From the projects we studied only Apache, Mozilla and Linux include code reviews as part of their development process. Mockus et. al found that Apache had a similar defect density than several commercial products. They argued that "fewer defects are injected into the [Apache's] code, or that other defect-finding activities such as inspections are conducted more frequently or more effectively." [17].

In Mozilla every contribution should be reviewed by at least two independent reviewers. The first type of review is conducted by the module owner or the module owner's peer (every module has an owner and a set of peers–individuals who are knowledgeable on that module). This review catches domain-specific problems. A patch that changes code in more than one module must receive a review from each module. The second type of review is called a super review. The goal of the super review is to find integration and infrastructural problems that may effect other modules or the user interface[10]. By requiring both types of reviews Mozilla ensures that someone with domain expertise and someone else with overall module and interface knowledge have approved the patch.

The Mozilla maintainers acknowledge that super reviews are a good way for "intermediary and advanced training", but "are a terrible mechanism for training in basic practices". The main concerns Mozilla maintainers have is that super-reviewers are very few and do not have the time to train other contributors: "[a] super-review [should] be the last stop for training." [23].

Over the years Apache has experimented with different types of code reviews. Apache currently uses a Commit-Review model, where core developers are allowed to commit changes that are then expected to be reviewed. The review takes place in the developers mailing list, an open environment where any contributor can participate. In an empirical study of code reviews on Apache we found that 9% of post-reviewed commits generated a discussion [21]. Apache post-commit reviews are not only useful as a way to find and eliminate defects, but because they happen in

7 http://guadec.org/
8 http://apachecon.com
9 http://conference.postgresql.org/

10 Mozilla's core review process requires the identification of individuals as module owners, module peers, and super reviewers. We can consider these as "knowledge" roles. Research is needed to understand how are these roles filled and who fills them.

a public forum (a mailing list) they also create awareness and diffuse knowledge, even to those who are not active participants in the review.

Code reviews practices in FOSS have started to influence industry. In [15] Lussier described how his company development process was (unexpectedly) influenced by their experiences participating in an FOSS community. Lussier was surprised that the code review practices of the Wine project resulted in better code, always ready to be released. His company decided to introduce a similar process in-house.

3. Storing and Retrieving Knowledge

As a software project evolves, a wealth of information is created (some automatically, some manually). Some of this information records communications between its contributors and users; other explains how the software system is evolving. We have previously demonstrated that historical records can be used to successfully reconstruct how a software system evolves [5].

In our research into the evolution of FOSS projects we have found that developers of mature FOSS projects value these records and ensure, often through policy, that these records be maintained; they form what Cubranic calls the "community memory" [2] of the project. In a study of historical records kept by FOSS communities we have observed the following types [7]:

- The source code itself. Version control systems allow developers to inspect the state of a file at any given time in the past, helping them understand how the system evolves. Source code sometimes is used as a communication medium, where notes and TODO lists are embedded as source code comments (such as the ones described in [25]).
- Defect tracking databases, such as Bugzilla, are frequently found in large FOSS projects. They provide a valuable source of information regarding defects (and their fixes) and feature requests.
- ChangeLogs are files that are usually updated when the system is changed, and provide a description of the given change. The Free Software Foundation requires all its projects to have a ChangeLog file. In those projects that have them, we have discovered that they are almost always properly updated [6].
- The Version Control logs of mature projects tend to have large, meaningful explanations. In the project Evolution, the average size of a log is 306 bytes, in Apache 1.3 it is 160 bytes, and in PostgreSQL it is 160 bytes, to cite just a few.
- Email is seen as an important source of discussion about the way software evolves.
- Code reviews are valuable discussions that provide good insight on why certain changes are performed the way they do. [21]. In contrast, version control logs and comments are shorter, usually omitting discussion of less satisfactory solutions. Having a link to a discussion might save the maintainer many hours in code comprehension and avoids time wasted trying to figure out why a given part of the system was implemented in a certain way.
- Documentation, including Web sites and wikis. FOSS projects are frequently using version control systems to store this type of information, which will allow contributors to inspect their state at any given date.

Some sources of information have a well defined format, such as version control logs and ChangeLogs, and are easy to correlate to lines of affected code. Correlating Bugzilla and source code is more difficult. It usually involves textual analysis of the description of the version control log. For example in [6], we describe regular expressions that were useful in the extraction of Bugzilla numbers from CVS commit logs. Correlating email messages is even more difficult. For Apache, we have been successful in creating automated and manual heuristics that help in the correlation of messages discussing code reviews [21]. Code reviews often involve *diffs* that contain the version in the repository against which the diff was made. However, general email discussions are much more difficult to correlate. Problems include determining the context of the discussion, reconstructing message threads, and resolving names to email addresses.

In [7] we proposed the concept of Evolutionary Annotations (EA), documentation that describes how a software system is evolving. EAs are information extracted (some automatically, some manually) from historical software development records. The purpose of evolutionary annotations is to explain why a project evolves in the way it does (contrary to documentation, that explains what the "current" system is doing). We proposed methods to retrieve them and correlate them to the source code, and described the design and implementation of a prototype for Eclipse that can filter and present these annotations alongside their corresponding source code.

3.1. How are communities using historical records?

Without controlled experiments it is difficult to determine how contributors use the historical information of a project, mainly because it is difficult to identify when a contributor access historical records, and for what purposes.

We have found, however, evidence that the information is being used by developers. Figure 1 shows excerpts from 2 email messages in which an question is answered by providing a link to older email discussions. One particular service that appears to be useful to contributors of FOSS is `Gmane.org`. `Gmane` is dedicated to provide three main services to FOSS mailing lists: archiving of messages (including permalinks), presenting email lists with a Web interface (including a blog-like option, and RSS feeds), and a powerful search engine. As the examples in figure 1 show the service provided by `Gmane` is being used by FOSS communities to retrieve and reuse the knowledge stored in their mailing lists.

The existence of tools intended to extract information from version control logs (such as Bonsai[11], cvschangelog[12], CVS History[13], CvsGraph[14], ViewVC[15] and many others) suggests that version control logs are useful to the developers. Unfortunately there have been no studies that try to understand how contributors (in both FOSS or proprietary systems) extract knowledge from version control repositories, in which circumstances it is useful, and how this extraction can be improved.

4. Is knowledge exchanged between communities?

One of the greatest assets that a FOSS project has is the size and diversity of its community.

In the proprietary software world knowledge is exchanged between organizations in very few ways. For example, a organization hires an employee from another organization, or by creating "knowledge exchange" contracts where an organization is willing to exchange its knowledge with another in exchange for some consideration. FOSS exchange of ideas and knowledge is often compared to that of science, where knowledge is created and exchanged without any requirement for compensation[16] [24].

Most FOSS communities have as their main goal the creation of a FOSS product, and the exchange and flow of knowledge and information is a side effect of it. While it is true that most FOSS projects have very small communities (one main contributor, with very few users), some communities have been able to achieve large numbers. The larger the community, the larger the pool of knowledge available to it. Even though most contributions come from few developers, any given knowledge contribution can have an important impact on the project. These contributions take many different forms: for example, pointers to sources of information (a person posts to a mailing lists a URL to knowledge stored by another project) or domain knowledge (in many cases users are more knowledgeable about the domain of an application than the core developers). Even just a note saying: "this program works great under 'such' operating system" might provide valuable knowledge.

FOSS projects are usually part of a larger FOSS ecology. They depend on other applications, and other applications might depend on them[17]. This creates a meta-community, where contributors and users from one community contribute (directly or indirectly) to the other communities. It is not uncommon for contributors of one project to subscribe to mailing lists in another project to gain awareness of where the project is and how it is evolving. In [22] Spinellis and Szypersky described how the Xine multimedia player[18] required 11 different libraries. Xine developers, therefore, required to know what these libraries did, and changes in these libraries would have had an effect on Xine, making them stakeholders (and users) in their development. Madey et al [16] used network analysis to demonstrate that FOSS projects create large clusters (a project is related to another project if they share at least one common contributor). They found that the largest cluster in Sourceforge connects 35% of its projects. Research is needed to find out if and how knowledge flows from one community to another via its common contributors.

11 `https://www.mozilla.org/bonsai.html`
12 `http://cvschangelog.sourceforge.net`
13 `http://cvshist.sf.net/`
14 `http://www.akhphd.au.dk/~bertho/cvsgraph/`
15 `http://www.viewvc.org/`

16 In recent years it is more frequent to find researchers who are opting for patenting their ideas before they make them public.
17 In some cases commercial applications are part of this ecology. `panotools` is a library and collection of applications that are used by some commercial applications (PTGui and PTassembler). These applications are very interested in fixing bugs and improving `panotools` and have contributed to its development; furthermore, the users of those commercial applications are indirect users of `panotools`–which benefits from their bug reports and suggestions.
18 `http://xinehq.de/`

```
[..]
> We switched physical mail servers and in transferring our ezmlm
> mailing lists and the vpopmail/qmailadmin installation ran into some
> problems. First, all mailing lists are freezing when receiving an
> e-mail from a subscriber with the following message in qmail-send
> log: "Sorry,_substitution_of_
> target_addresses_into_headers_with_#A#>_or_#T#>_is_unsafe_and_not_permitted./"

This thread may help you :
http://article.gmane.org/gmane.mail.ezmlm/4297
http://article.gmane.org/gmane.mail.ezmlm/4298
[..]
```

```
[..]
> I have seen several posts for this but none resolve my issue.

You haven't been looking hard enough ;-)
http://www.lowagie.com/iText/faq.html#jsp
http://article.gmane.org/gmane.comp.java.lib.itext.general/8850
[..]
```

Figure 1. Excepts from mail messages that reference older discussions.

4.1. The Slashdot Effect

Blogs have an influence in the exchange of knowledge among the FOSS communities. Slashdot (slashdot.org, "News for Nerds, Stuff that Matters") is a blog dedicated to the discussion of technology news, particularly those of interest to FOSS communities [19]. News entries are usually submitted by readers. Its infrastructure enables readers to post comments to the entries and to rank those submitted by their peers (in an effort to improve the signal-to-noise ratio of comments). A special section called "Ask Slashdot" invites readers to submit questions that might be of general interest, expecting other readers to post answers to the questions. Slashdot conducts interviews and publishes book reviews too.

Slashdot provides a place where members of different FOSS communities can gather and discuss issues that might be of their interest. Slashdot is particularly useful to provide awareness, e.g. what other FOSS communities are doing, and issues that affect FOSS in general. It also serves as a place to advertise important advancements in a FOSS project.

Another site worth mentioning is Groklaw[19]. Groklaw specializes in the discussion of intellectual property law and its effects on the FOSS world. It was formed in 2003 as a response to the legal challenges brought by SCO against IBM and other organizations with regard to some intellectual property found in the Linux Kernel (for an overview of the legal case see [4]). Groklaw's model is very similar to Slashdot's, although it maintains stronger editorial control, resulting in higher quality of entries and comments. Groklaw demonstrates that a Web site can link two different communities (in this case typical FOSS contributors and legal experts) to create an environment where knowledge is shared, exchanged and enhanced between them. This is encapsulated in a comment by its creator, Pamela Jones: "Some of the volunteers knew things I didn't, especially about the code issues, but they didn't realize what they knew was useful legally. [...] People are hungry to understand legal news, and they want to help." [11].

19 `http://groklaw.net`

5. Discussion and Future Work

This paper reports preliminary results on how knowledge flows in FOSS communities. We have found that FOSS communities have developed multiple methods to communicate and exchange knowledge. None of the projects surveyed has exactly the same methods. This can be due to one or several factors: for example, their application domains are different; or their communities work better with different methods.

We require a lot of research in this area. We need to perform quantitative empirical studies on how FOSS projects generate, share, store and retrieve knowledge. We also need to perform controlled experiments to compare methods and to understand their advantages and disadvantages, and under which scenarios they can be useful. The results from these studies can help FOSS communities to select the methods better suited for their particular needs. We need to explore new methods to exchange and store knowledge, and equally important, how to make it easier to find knowledge (either who has it, or by finding and retrieving it).

Acknowledgments

The author would like to thank the anonymous reviewers for their suggestions to improve this paper. This research is supported by the National Sciences and Engineering Research Council of Canada.

References

[1] R. Bowen. A day in the life of #apache. O'Reilly ONLamp.com Apache DevCenter, 2003-2005. Montly column.

[2] D. Cubranić, G. C. Murphy, J. Singer, and K. S. Booth. Learning from project history: A case study for software development. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 82–91, 2004.

[3] M. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751, 1986.

[4] L. Geppert. Battle of the Xs. *Spectrum*, 40(8):16–17, Aug. 2003.

[5] D. M. German. Using software trails to reconstruct the evolution of software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):367–384, 2004.

[6] D. M. German. An empirical study of fine-grained software modifications. *Journal of Empirical Software Engineering*, 2005. Accepted for publication Sept 25, 2005, to appear in the Special Issue of Best Papers of ICSM 2004.

[7] D. M. German, P. Rigby, and M. A. Storey. Using Evolutionary Annotations from Change Logs to enhance Program Comprehension. In *3rd International Workshop on Mining Software Repositories (MSR 2006)*, May 2005.

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

[9] IBM Corporation. IBM Statement of Non-Assertion of Named Patents Against OSS. `http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf`, Jan. 2001.

[10] P. M. Johnson and D. Tjahjono. Does every inspection really need a meeting? *Journal of Empirical Software Engineering*, 5(3):9–35, 1998.

[11] P. Jones. EOF: open legal research. *Linux J.*, 2004(121):13, 2004.

[12] B. Kogut and A. Metiu. Open-source software development and distributed innovation. *Oxford Review of Economic Policy*, 17(2):258–264, 2001.

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

[14] K. R. Lakhani and B. Wolf. *Perspectives on Free and Open Source Software*, chapter Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, pages 3–21. MIT Press, 2005.

[15] S. Lussier. New tricks: How open source changed the way my team works. *IEEE Software*, 21(1):68–72, 2004.

[16] G. Madey, V. Freeh, and R. Tynan. *Free/Open Source Software Development*, chapter Modeling the F/OSS Community: A Quantitative Investigation, in Free/Open Source Software Development. Idea Publishing, 2004.

[17] A. Mockus, R. T. Fielding, and J. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1–38, July 2002.

[18] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution patterns of open-source software systems and communities. In *IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution*, pages 76–85, New York, NY, USA, 2002. ACM Press.

[19] N. Poor. Mechanisms of an online public sphere: The website Slashdot. *Journal of Computer-Mediated Communication*, 10(2), 2005.

[20] E. Raymond. *The Cathedral & the Bazaar.* O'Reilly, 1999.

[21] P. Rigby and D. M. German. A preliminary examination of code review processes in open source projects. Technical Report DCS -305-IR, University of Victoria, 2006.

[22] D. Spinellis and C. Szypersky. How is Open Source Software Affecting Software Development. *IEEE Software*, 21(1):28–33, Jan-Feb 2004.

[23] The Mozilla Foundation. Frequently Asked Questions about mozilla.org's Code Review Process. http://www.mozilla.org/hacking/code-review-faq.html, June 2006.

[24] J. Willinsky. Unacknowledged convergence of open source, open access, and open science. *First Monday*, 10(8), August 2005.

[25] A. T. T. Ying, J. L. Wright, and S. Abrams. Source code that talks: an exploration of eclipse task comments and their implication to repository mining. In *MSR '05: Proceedings of the 2005 International Workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM Press.