Free/Open Source Software Development: Recent Research Results and Methods

Walt Scacchi
Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697-3425 USA
+1-949-824-4130 (v), +1-949-824-1715 (f)
Wscacchi@uci.edu

27 August 2006
Revised version to appear in:
M.V. Zelkowitz (ed.), *Advances in Computers*, Vol. 69, 2007

Abstract

The focus of this chapter is to review what is known about free and open source software development (FOSSD) work practices, development processes, project and community dynamics, and other socio-technical relationships. It does not focus on specific properties or technical attributes of different FOSS systems, but it does seek to explore how FOSS is developed and evolved. The chapter provides a brief background on what FOSS is and how free software and open source software development efforts are similar and different. From there attention shifts to an extensive review of a set of empirical studies of FOSSD that articulate different levels of analysis. These characterize what has been analyzed in FOSSD studies across levels that examine why individuals participate; resources and capabilities supporting development activities; how cooperation, coordination, and control are realized in projects; alliance formation and inter-project social networking; FOSS as a multi-project software ecosystem, and FOSS as a social movement. Following this, the chapter reviews how different research methods are employed to examine different issues in FOSSD. These include reflective practice and industry polls, survey research, ethnographic studies, mining FOSS repositories, and multi-modal modeling and analysis of FOSSD processes and socio-technical networks. Finally, there is a discussion of limitations and constraints in the FOSSD studies so far, attention to emerging opportunities for future FOSSD studies, and then conclusions about what is known about FOSSD through the empirical studies reviewed here.

Keywords: Free software, Open source software, empirical studies, socio-technical relationships

Introduction

This chapter examines and compares practices, patterns, and processes that emerge in empirical studies of free/open source software development (FOSSD) projects. FOSSD is a way for building, deploying, and sustaining large software systems on a global basis, and differs in many interesting ways from the principles and practices traditionally advocated for software engineering [Somerville 2004]. Hundreds of FOSS systems are now in use by thousands to millions of end-users, and some of these FOSS systems entail hundreds-of-thousands to millions of lines of source code. So what's going on here, and how are FOSSD processes that are being used to build and sustain these projects different, and how might differences be employed to explain what's going on with FOSSD, and why.

One of the more significant features of FOSSD is the formation and enactment of complex software development processes and practices performed by loosely coordinated software developers and contributors. These people may volunteer their time and skill to such effort, and may only work at their personal discretion rather than as assigned and scheduled. However, increasingly, software developers are being assigned as part of the job to develop or support FOSS systems, and thus to become involved with FOSSD efforts. Further, FOSS developers are generally expected (or prefer) to provide their own computing resources (e.g., laptop computers on the go, or desktop computers at home), and bring their own software development tools with them. Similarly, FOSS developers work on software projects that do not typically have a corporate owner or management staff to organize, direct, monitor, and improve the software development processes being put into practice on such projects. But how are successful FOSSD projects and processes possible without regularly employed and scheduled software development staff, or without an explicit regime for software engineering project management? What motivates software developers participate in FOSSD projects? Why and how are large FOSSD projects sustained? How are large FOSSD projects coordinated, controlled or managed without a traditional project management team? Why and how might these answers to these questions change over time? These are the kinds of questions that will be addressed in this article.

The remainder of this chapter is organized as follows. The next section provides a brief background on what FOSS is and how free software and open source software development efforts are similar and different. From there attention shifts to an extensive review of a set of empirical studies of FOSSD that articulate different levels of analysis. Following this, the chapter reviews how different research methods are employed to examine different issues in FOSSD. Finally, there is a discussion of limitations and constraints in the FOSSD studies so far, attention to emerging opportunities for future FOSSD studies, and then conclusions about what is known about FOSSD through the empirical studies reviewed here.

What is free/open source software development?

Free (as in freedom) software and open source software are often treated as the same thing [Feller and Fitzgerald 2002, Feller, *et al.* 2005, Koch 2005]. However, there are differences between them with regards to the licenses assigned to the respective software.

Free software generally appears licensed with the GNU General Public License (GPL), while OSS may use either the GPL or some other license that allows for the integration of software that may not be free software. Free software is a social movement [cf. Elliott and Scacchi 2006], whereas OSSD is a software development methodology, according to free software advocates like Richard Stallman and the Free Software Foundation [Gay 2002]. Yet some analysts also see OSS as a social movement distinct from but related to the free software movement. The hallmark of free software and most OSS is that the source code is available for remote access, open to study and modification, and available for redistribution to other with few constraints, except the right to insure these freedoms. OSS sometimes adds or removes similar freedoms or copyright privileges depending on which OSS copyright and end-user license agreement is associated with a particular OSS code base. More simply, free software is always available as OSS, but OSS is not always free software[1]. This is why it often is appropriate to refer to FOSS or FLOSS (L for *Libre*, where the alternative term "libre software" has popularity in some parts of the world) in order to accommodate two similar or often indistinguishable approaches to software development. Subsequently, for the purposes of this article, focus is directed at FOSSD practices, processes, and dynamics, rather than to software licenses though such licenses may impinge on them. However, when appropriate, particular studies examined in this review may be framed in terms specific to either free software or OSS when such differentiation is warranted.

FOSSD is mostly not about software engineering, at least not as SE is portrayed in modern SE textbooks [cf. Sommerville 2004]. FOSSD is not SE done poorly. It is instead

[1] Thus at times it may be appropriate to distinguish conditions or events that are generally associated or specific to either free software development or OSSD, but not both.

a different approach to the development of software systems where much of the development activity is openly visible, and development artifacts are publicly available over the Web. Furthermore, substantial FOSSD effort is directed at enabling and facilitating social interaction among developers (and sometimes also end-users), but generally there is no traditional software engineering project management regime, budget or schedule. FOSSD is also oriented towards the joint development of an ongoing community of developers and users concomitant with the FOSS system of interest.

FOSS developers are typically also end-users of the FOSS they develop, and other end-users often participate in and contribute to FOSSD efforts. There is also widespread recognition that FOSSD projects can produce high quality and sustainable software systems that can be used by thousands to millions of end-users [Mockus, Fielding, Herbsleb 2002]. Thus, it is reasonable to assume that FOSSD processes are not necessarily of the same type, kind, or form found in modern SE projects [cf. Sommerville 2004]. While such approaches might be used within an SE project, there is no basis found in the principles of SE laid out in textbooks that would suggest SE projects typically adopt or should practice FOSSD methods. Subsequently, what is known about SE processes, or modeling and simulating SE processes, may not be equally applicable to FOSSD processes without some explicit rationale or empirical justification. Thus, it is appropriate to survey what is known so far about FOSSD.

Results from recent studies of FOSSD

There are a growing number of studies that offer some insight or findings on FOSSD practices each in turn reflects on different kinds of processes that are not well understood at this time. The focus in this chapter is directed to empirical studies of FOSSD projects using small/large research samples and analytical methods drawn from different academic disciplines. Many additional studies of FOSS can be found within a number of Web portals for research papers that empirically or theoretically examine FOSSD projects. Among them are those at MIT FOSS research community portal (opensource.mit.edu) with 200 or so papers already contributed, and also at Cork College in Ireland (opensource.ucc.ie) which features links to multiple special issue journals and proceedings from international workshops of FOSS research. Rather than attempt to survey the complete universe of studies in these collections, the choice instead is to sample a smaller set of studies that raise interesting issues or challenging problems for understanding what affects how FOSSD efforts are accomplished, as what kinds of socio-technical relationships emerge along the way to facilitate these efforts.

One important qualifier to recognize is that the studies below generally examined carefully identified FOSSD projects or a sample of projects, so the results presented should not be assumed to apply to all FOSSD projects, or to projects that have not been studied. Furthermore, it is important to recognize that FOSSD is no silver bullet that resolves the software crisis. Instead it is fair to recognize that most of the nearly 130,000 FOSSD projects associated with Web portals like SourceForce.org have very small teams of two or less developers [Madey *et al.* 2002, 2005], and many projects are inactive or have yet to release any operational software. However, there are now at least a few thousand FOSSD projects that are viable and ongoing. Thus, there is a sufficient universe of diverse FOSSD projects to investigate, analyze, and compare in the course of moving towards an articulate and empirically grounded theory or model of FOSSD.

Consequently, consider the research findings reported or studies cited below as starting points for further investigation, rather than as defining characteristics of most or all FOSSD projects or processes.

Attention now shifts to an extensive review of a sample of empirical studies of FOSSD that are grouped according to different levels of analysis. These characterize what has been analyzed in FOSSD studies across levels that examine why individuals participate in FOSSD efforts; what resources and capabilities shared by individuals and groups developing FOSS; projects as organizational form for cooperating, coordinating, and controlling FOSS development effort; alliance formation and inter-project social networking; FOSS as a multi-project software ecosystem, and FOSS as a social movement. These levels thus span the study of FOSSD from individual participant to social world. Each level is presented in turn. Along the way, figures from FOSSD studies or data exhibits collected from FOSSD projects will be employed to help illustrate concepts described in the studies under review.

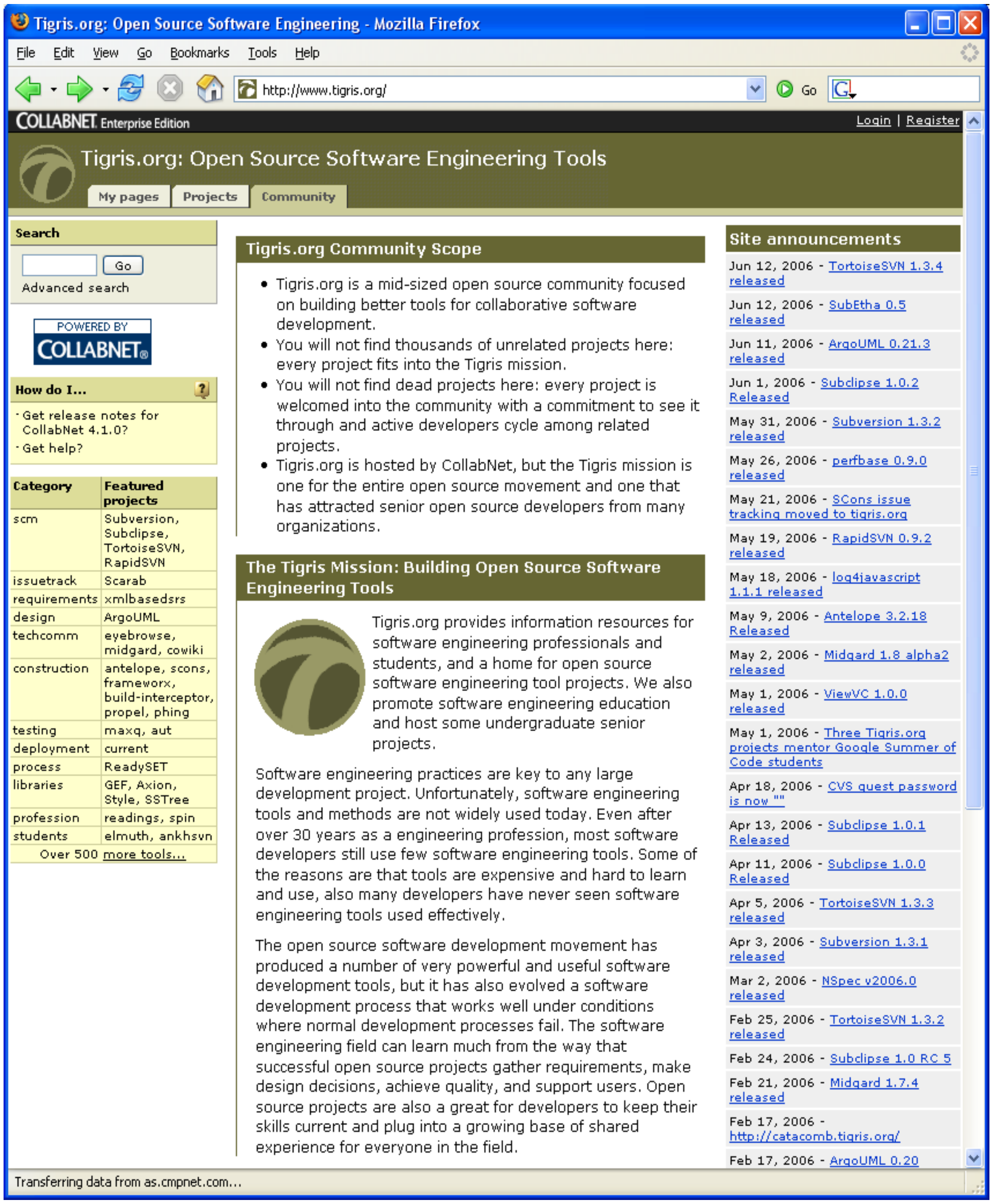
Individual Participation in FOSSD Projects

One of the most common questions about FOSSD projects to date is why will software developers join and participate in such efforts, often without pay for sustained periods of time. A number of surveys of FOSS developers [FLOSS 2002, Ghosh and Prakash 2000, Lakhani, *et al.* 2002, Hars and Ou 2002, Hann, *et al.* 2002, Hertel, *et al.* 2003] has posed such questions, and the findings reveal the following.

There are complex motivations for why FOSS developers are willing to allocate their time, skill, and effort by joining a FOSS project [Hars and Ou 2002, Hertel, *et al.* 2003 von Krogh, *et al*. 2003]. Sometimes they may simply see their effort as something that is fun, personally rewarding, or provides a venue where they can exercise and improve their technical competence in a manner that may not be possible within their current job or line of work [Crowston and Scozzi 2002]. However, people who participate, contribute, and join FOSS projects tend to act in ways where building trust and reputation [Stewart and Gosain 2001], achieving "geek fame" [Pavlicek 2000], being creative [Fischer 2001], as well as giving and being generous with one's time, expertise, and source code [Bergquist and Ljungberg 2001] are valued traits. In the case of FOSS for software engineering design systems, participating in such a project is a viable way to maintain or improve software development skills, as indicated in Exhibit 1 drawn from the Tigris.org open source software engineering community portal.

Becoming a central actor (or node) in a social network of software developers that interconnects multiple FOSS projects is also a way to accumulate social capital and recognition from peers. One study reports that 60% or more FOSS developers participate in two or more projects, and on the order of 5% participate in 10 or more FOSS projects [Hars and Ou 2002]. However, the vast majority of source code that becomes part of FOSS released by a project is typically developed by a small group of core developers who control the architecture and direction of development [cf. Mockus, Fielding, and Herbsleb 2002]. Subsequently, most participants typically contribute to just a single module, though a small minority of modules may be include patches or modifications contributed by hundreds of contributors [Ghosh and Prakash 2000]. In addition, participation in FOSS projects as a core developer can realize financial rewards in terms of higher salaries for conventional software development jobs [Hann 2002, Lerner 2002]. However, it also enables the merger of independent FOSS systems into larger composite ones that gain the critical mass of core developers to grow more substantially and attract ever larger user-developer communities [Madey, *et al.* 2005, Scacchi 2005].

People who participate in FOSS projects do so within one or more roles. Classifications of the hierarchy of roles that people take and common tasks they perform when participating in a FOSS project continue to appear [Crowston and Howison 2006, Gacek and Arief 2004, Jensen and Scacchi 2006b, Ye and Kishida 2003] . Exhibit 2 from the Object-Oriented Graphics Rendering Engine (OGRE) project provides a textual description of the principal roles (or “levels”) in that project community.


Tigris.org: Open Source Software Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.tigris.org/
Go
COLLABNET Enterprise Edition
Login | Register
My pages
Projects
Community
Go
POWERED BY COLLABNET
Transferring data from as.cmpnet.com...

Tigris.org: Open Source Software Engineering Tools

Search

Advanced search

How do I...

- Get release notes for CollabNet 4.1.0?
- Get help?

Category	Featured projects
scm	Subversion, Subclipse, TortoiseSVN, RapidSVN
issuetrack	Scarab
requirements	xmlbasedsrs
design	ArgoUML
techcomm	eyebrowse, midgard, cowiki
construction	antelope, scons, frameworx, build-interceptor, propel, phing
testing	maxq, aut
deployment	current
process	ReadySET
libraries	GEF, Axion, Style, SSTree
profession	readings, spin
students	elmuth, ankhsvn
Over 500	more tools...

Tigris.org Community Scope

- Tigris.org is a mid-sized open source community focused on building better tools for collaborative software development.
- You will not find thousands of unrelated projects here: every project fits into the Tigris mission.
- You will not find dead projects here: every project is welcomed into the community with a commitment to see it through and active developers cycle among related projects.
- Tigris.org is hosted by CollabNet, but the Tigris mission is one for the entire open source movement and one that has attracted senior open source developers from many organizations.

The Tigris Mission: Building Open Source Software Engineering Tools

Tigris.org provides information resources for software engineering professionals and students, and a home for open source software engineering tool projects. We also promote software engineering education and host some undergraduate senior projects.

Software engineering practices are key to any large development project. Unfortunately, software engineering tools and methods are not widely used today. Even after over 30 years as a engineering profession, most software developers still use few software engineering tools. Some of the reasons are that tools are expensive and hard to learn and use, also many developers have never seen software engineering tools used effectively.

The open source software development movement has produced a number of very powerful and useful software development tools, but it has also evolved a software development process that works well under conditions where normal development processes fail. The software engineering field can learn much from the way that successful open source projects gather requirements, make design decisions, achieve quality, and support users. Open source projects are also a great for developers to keep their skills current and plug into a growing base of shared experience for everyone in the field.

Site announcements

Jun 12, 2006 - TortoiseSVN 1.3.4 released

Jun 12, 2006 - SubEtha 0.5 released

Jun 11, 2006 - ArgoUML 0.21.3 released

Jun 1, 2006 - Subclipse 1.0.2 Released

May 31, 2006 - Subversion 1.3.2 released

May 26, 2006 - perfbase 0.9.0 released

May 21, 2006 - SCons issue tracking moved to tigris.org

May 19, 2006 - RapidSVN 0.9.2 released

May 18, 2006 - log4javascript 1.1.1 released

May 9, 2006 - Antelope 3.2.18 Released

May 2, 2006 - Midgard 1.8 alpha2 released

May 1, 2006 - ViewVC 1.0.0 released

May 1, 2006 - Three Tigris.org projects mentor Google Summer of Code students

Apr 18, 2006 - CVS guest password is now ""

Apr 13, 2006 - Subclipse 1.0.1 Released

Apr 11, 2006 - Subclipse 1.0.0 Released

Apr 5, 2006 - TortoiseSVN 1.3.3 released

Apr 3, 2006 - Subversion 1.3.1 released

Mar 2, 2006 - NSpec v2006.0 released

Feb 25, 2006 - TortoiseSVN 1.3.2 released

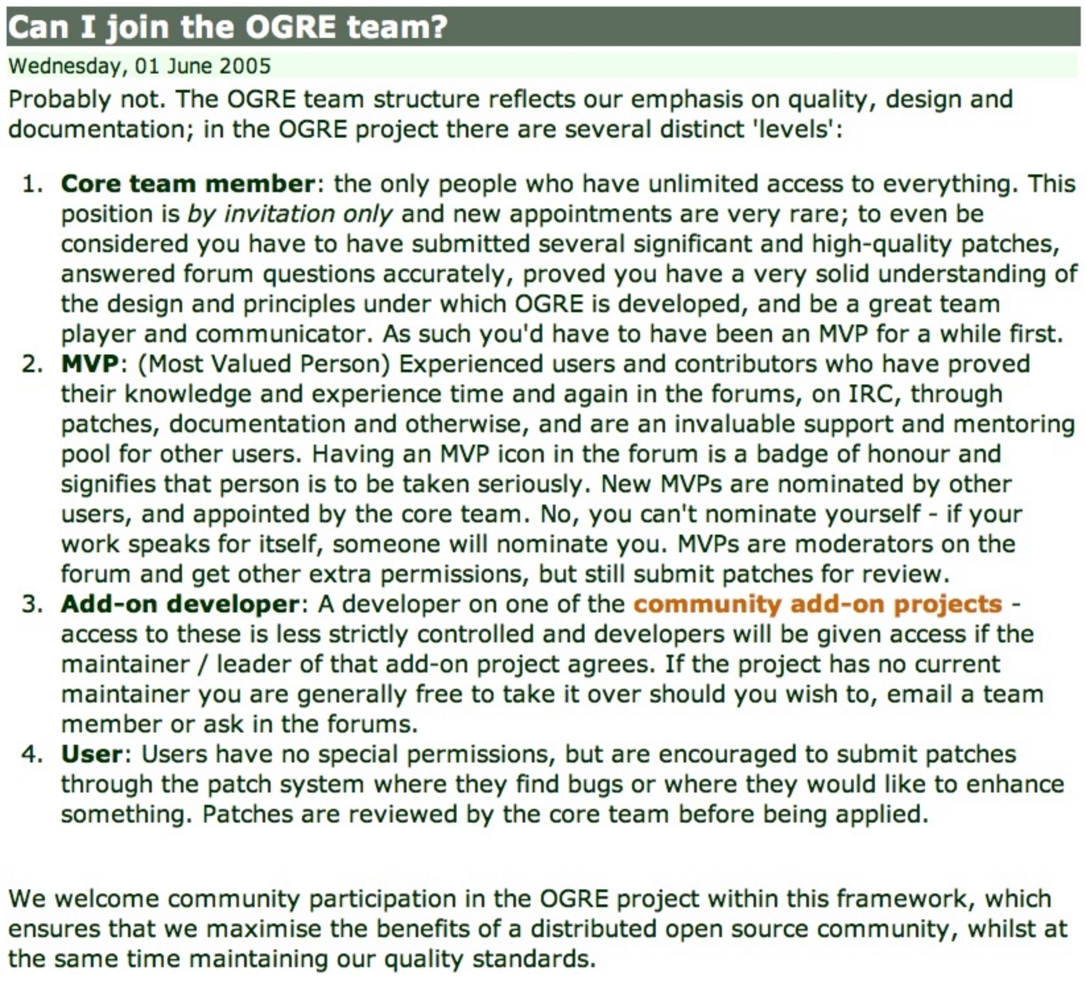
Feb 24, 2006 - Subclipse 1.0 RC 5

Feb 21, 2006 - Midgard 1.7.4 released

Feb 17, 2006 - http://catacomb.tigris.org/

Feb 17, 2006 - ArgoUML 0.20

Exhibit 1. An example in the bottom paragraph highlighting career/skill development opportunities that encourage participation in FOSSD projects (source: http://www.tigris.org, June 2006)



Can I join the OGRE team?

Wednesday, 01 June 2005

Probably not. The OGRE team structure reflects our emphasis on quality, design and documentation; in the OGRE project there are several distinct 'levels':

1. **Core team member**: the only people who have unlimited access to everything. This position is *by invitation only* and new appointments are very rare; to even be considered you have to have submitted several significant and high-quality patches, answered forum questions accurately, proved you have a very solid understanding of the design and principles under which OGRE is developed, and be a great team player and communicator. As such you'd have to have been an MVP for a while first.
2. **MVP**: (Most Valued Person) Experienced users and contributors who have proved their knowledge and experience time and again in the forums, on IRC, through patches, documentation and otherwise, and are an invaluable support and mentoring pool for other users. Having an MVP icon in the forum is a badge of honour and signifies that person is to be taken seriously. New MVPs are nominated by other users, and appointed by the core team. No, you can't nominate yourself - if your work speaks for itself, someone will nominate you. MVPs are moderators on the forum and get other extra permissions, but still submit patches for review.
3. **Add-on developer**: A developer on one of the **community add-on projects** - access to these is less strictly controlled and developers will be given access if the maintainer / leader of that add-on project agrees. If the project has no current maintainer you are generally free to take it over should you wish to, email a team member or ask in the forums.
4. **User**: Users have no special permissions, but are encouraged to submit patches through the patch system where they find bugs or where they would like to enhance something. Patches are reviewed by the core team before being applied.

We welcome community participation in the OGRE project within this framework, which ensures that we maximise the benefits of a distributed open source community, whilst at the same time maintaining our quality standards.

Exhibit 2. Joining the OGRE FOSS development team by roles/level. (Source: http://www.ogre3d.org/index.php?option=com_content&task=view&id=333&Itemid=87 June 2005)

Typically, it appears that people join a project and specialize in a role (or multiple roles) they find personally comfortable and intrinsically motivating [von Krogh *et al.* 2004]. In contrast to traditional software development projects, there is no explicit assignment of developers to roles, though individual FOSSD projects often post guidelines or "help wanted here" for what roles for potential contributors are in greatest need. Exhibit 3 provides an example drawn from popular FOSS mpeg-2 video player, the VideoLan Client (VLC).

VideoLAN needs your help
2006-06-19

There are many things that we would like to improve in VLC, but that we don't, because we simply don't have enough time. That's why we are currently looking for some help. We have identified several small projects that prospective developers could work on. Knowledge of C and/or C++ programming will certainly be useful, but you don't need to be an expert, nor a video expert. Existing VLC developers will be able to help you on these projects. You can find the list and some instructions on the dedicated Wiki page. Don't hesitate to join us on IRC or on the mailing-lists. We are waiting for you!

Exhibit 3. An example request for new FOSS developers to come forward and contribute their assistance to developing more functionality to the VLC system.
(source: http://www.videolan.org/, June 2006)

It is common in FOSS projects to find end-users becoming contributors or developers, and developers acting as end-users [Mockus, Fielding and Herbsleb 2002, Nakakoji *et al.* 2002, Scacchi 2002, von Hippel and von Krogh 2003]. As most FOSS developers are themselves end-users of the software systems they build, they may have an occupational incentive and vested interest in making sure their systems are really useful. However the vast majority of participants probably simply prefer to be users of FOSS systems, unless or until their usage motivates them to act through some sort of contribution. Avid users with sufficient technical skills may actually work their way up (or "level up") through each of the roles and eventually become a core developer (or "elder"), as suggested by Figure 1. As a consequence, participants within FOSS project often participate in different roles within both technical and social networks [Lopez-Fernandez, *et al.* 2004, 2006, Preece 2000, Scacchi 2005, Smith and Pollock1999] in the course of developing, using, and evolving FOSS systems.

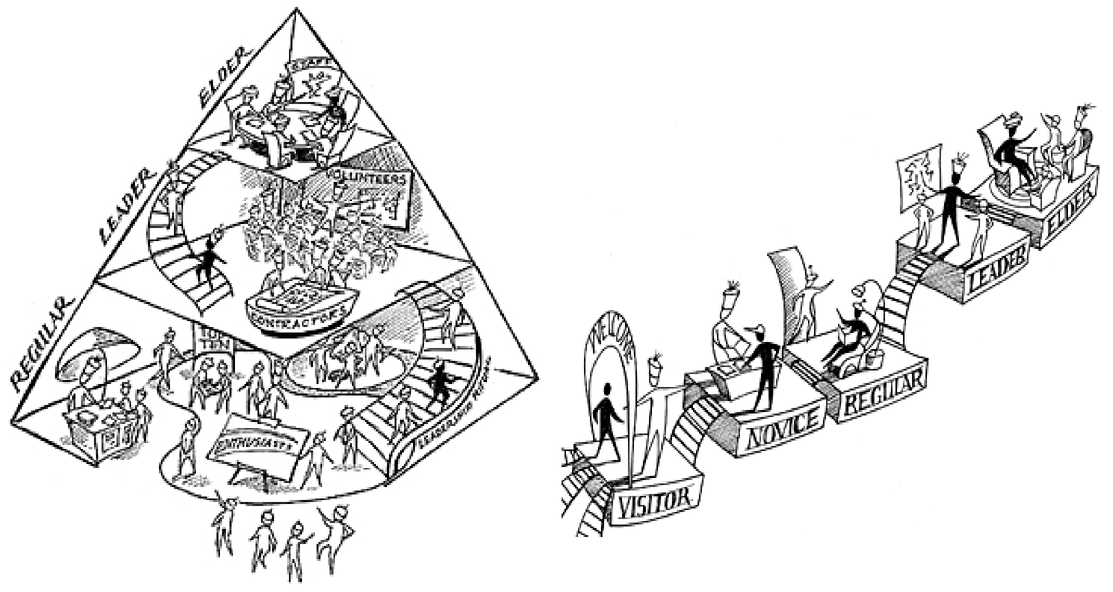


Figure 1. A visual depiction of role hierarchy within a project community (source: Kim [2000]).

Making contributions is often a prerequisite for advancing technically and socially within an ongoing project, as is being recognized by other project members as having made substantive contributions [Fielding 1999, Kim 2000]. Most commonly, FOSS project participants contribute their time, skill and effort to modify or create different types of software representations or content (source code, bug reports, design diagrams, execution scripts, code reviews, test case data, Web pages, email comments, online chat, etc.; also collectively called "software informalisms" [Scacchi 2002]) to Web sites of the FOSS projects they join. The contribution—the authoring, hypertext linking (when needed), and posting/uploading—of different types of content helps to constitute an ecology of document genres [Erickson 2000, Spinuzzi 2000] that is specific to a FOSS project, though individual content types are widely used across most FOSS projects. Similarly, the particular mix of online documents employed by participants on a FOSS project articulates an information infrastructure for framing and solving problems that arise in the ongoing development, deployment, use, and support of the FOSS system at the center of a project.

Administrators of FOSS project Web sites and source code repositories serve as gatekeepers in the choices they make for what information to post, when and where within the site to post it, as well as what not to post [cf. Hakken 1999, Hine 2000, Smith and Pollock 1999]. Similarly, they may choose to create a site map that constitutes a classification of site and domain content, as well as outlining community structure and boundaries.

Most frequently, participants in FOSS projects engage in online discussion forums or threaded email messages as a central way to observe, participate in, and contribute to public discussions of topics of interest to ongoing project participants [Yamauchi 2000]. However, these people also engage in private online or offline discussions that do not get posted or publicly disclosed, due to their perceived sensitive content.

FOSS developers generally find the greatest benefit from participation is the opportunity to learn and share what they know about software system functionality, design, methods, tools, and practices associated with specific projects or project leaders [FLOSS 2002, Ghosh and Prakash 2000, Lakhani *et al.* 2002]. FOSSD is a venue for learning for individuals, project groups, and organizations, and learning organizations are ones which can continuously improve or adapt their processes and practices [Huntley 2003, Ye and Kishida 2003]. However, though much of the development work in FOSSD projects is unpaid or volunteer, individual FOSS developers often benefit with higher average wages and better employment opportunities (at present), compared to their peers lacking FOSSD experience or skill [Hann *et al.* 2002, Lerner and Tirole 2002].

Consequently, how and why software developers will join, participate in, and contribute to an FOSSD project seems to represent a new kind of process affecting how FOSS is developed and maintained [cf. Bonaccorsi and Rossi 2006, Jensen and Scacchi 2006b, Scacchi 2005, von Krogh, Spaeth and Lakhani 2003]. Subsequently, discovering, observing, modeling, analyzing, and simulating what this process is, how it operates, and how it affects software development is an open research challenge for the software process research community.

Studies have also observed and identified the many roles that participants in an FOSSD project perform [Gacek and Arief 2004, Jensen and Scacchi 2006b,Ye and Kishida 2003]. These roles are used to help explain who does what, which serves as a precursor to explanations of how FOSSD practices or processes are accomplished and hierarchically arrayed. However such a division of labor is dynamic, not static or fixed. This means that participants can move through different roles throughout the course of a project over time, depending on their interest, commitment, and technical skill (as suggested in Figure 1). Typically, participants start at the periphery of a project in the role of end-user by downloading and using the FOSS associated with the project. They can then move into roles like bug-reporter, code reviewer, code/patch contributor, module owner (development coordinator), and eventually to core developer or project leader. Moving through these roles requires effort, and the passage requires being recognized by other participants as a trustworthy and accomplished contributor.

Role-task migration can and does arise within FOSSD projects, as well as across projects [Jensen and Scacchi 2006b]. Social networking, software sharing, and project internetworking enables this. But how do role-task migration processes or trajectories facilitate or constrain how FOSSD occurs? Role-task migration does not appear as a topic addressed in traditional SE textbooks or studies (see Sim and Holt [1998] for a notable exception), yet it seems to be a common observation in FOSSD projects. Thus, it seems that discovery, modeling, simulating or re-enacting [cf. Jensen and Scacchi 2006a] how individual developers participate in a FOSSD effort while enacting the role-task migration process, and how it affects or contributes to other software development or quality assurance processes, is an area requiring further investigation.

Resources and Capabilities Supporting FOSSD

What kinds of resources or development capabilities are needed to help make FOSS efforts more likely to succeed? Based on what has been observed and reported across many empirical studies of FOSSD projects, the following kinds of socio-technical resources enable the development of both FOSS software and ongoing project that is sustaining its evolution, application and refinement, though other kinds of resources may also be involved [Scacchi 2002, 2005, 2006b].

Personal software development tools and networking support

FOSS developers, end-users, and other volunteers often provide their own personal computing resources in order to access or participate in a FOSS development project. They similarly provide their own access to the Internet, and may even host personal Web sites or information repositories. Furthermore, FOSS developers bring their own choice of tools and development methods to a project. Sustained commitment of personal resources helps *subsidize* the emergence and evolution of the ongoing project, its shared (public) information artifacts, and resulting open source code. It spreads the cost for creating and maintaining the information infrastructure of the virtual organization that constitute a FOSSD project [Crowston and Scozzi 2002, Elliott and Scacchi 2005, Noll and Scacchi 1999]. These in turn help create recognizable shares of the FOSS commons [cf. Benkler 2006, Ghosh 2005, Lessig 2005, Ostrom, Calvert and Eggertsson 1990] that are linked (via hardware, software, and Web) to the project's information infrastructure.

Beliefs supporting FOSS Development

Why do software developers and others contribute their skill, time, and effort to the development of FOSS and related information resources? Though there are probably many diverse answers to such a question, it seems that one such answer must account for the belief in the freedom to access, study, modify, redistribute and share the evolving results from a FOSS development project. Without such belief, it seems unlikely that there could be "free" and "open source" software development projects [DiBona, *et al*. 1999, 2005, Fogel 2005, Gay 2002, Pavlicek 2000, Williams 2002]. However, one important consideration that follows is what the consequences from such belief are, and how these consequences are put into action.

In a longitudinal study of the free software project GNUenterprise.org, Elliott and Scacchi [2003, 2005, 2006] identified many kinds of beliefs, values, and social norms that shaped actions taken and choices made in the development of the GNUe software. Primary among them were *freedom of expression* and *freedom of choice*. Neither of these freedoms is explicitly declared, assured, or protected by free software copyright or commons-based intellectual property rights, or end-user license agreements (EULAs)[2]. However, they are central tenets free or open source modes of production and culture [Benkler 2006, Ghosh 2005, Lessig 2005]. In particular, in FOSS projects like GNUenterprise.org and others, these additional freedoms are expressed in choices for what to develop or work on (e.g., choice of work subject or personal interest over work assignment), how to develop it (choice of method to use instead of a corporate standard), and what tools to employ (choice over which personal tools to employ versus only using what is provided). They also are expressed in choices for when to release work products (choice of satisfaction of work quality over schedule), determining what to review and when (modulated by ongoing project ownership responsibility), and expressing what can be said to whom with or without reservation (modulated by trust and accountability mechanisms). Shared belief and practice in these freedoms of expression and choice are part of the virtual organizational culture that characterizes a FOSSD project like GNUenterprise.org [Elliott and Scacchi 2005]. Subsequently, putting these beliefs and cultural resources into action continues to build and reproduce socio-technical interaction networks that enabled sustained FOSSD projects and free software.

[2] EULAs associated with probably all software often seek to declare "freedom from liability" from people who want to use licensed software for intended or unintended purposes. But a belief in liability freedom is not the focus here.

Limitations and Constraints for FOSS Research

FOSSD is certainly not a panacea for developing complex software systems, nor is it simply software engineering done poorly. Instead, it represents an alternative community-intensive socio-technical approach to develop software systems, artifacts, and social relationships. However, it is not without its limitations and constraints. Thus, we should be able to help see these limits as manifest within the level of analysis or research for empirical FOSSD studies examined above.

First, in terms of participating, joining, and contributing to FOSS projects, an individual developer’s interest, motivation, and commitment to a project and its contributors is dynamic and not indefinite. FOSS developers are loathe to find themselves contributing to a project that is realizing commercial or financial benefits that are not available to all contributors, or that are concentrated to benefit a particular company, again without some share going to the contributors. Some form of reciprocity seems necessary to sustain participation, whereas a perception of exploitation by others can quickly dissolve a participant’s commitment to further contribute, or worse to dissuade other participants to abandon an open source project that has gone astray. If linchpin developers lose interest, then unless another contributor comes forward to fill in or take over role and responsibility for the communication and coordination activities of such key developers, then the FOSS system may quickly become brittle, fragile, and difficult to maintain. Thus, participation, joining, and contributing must become sustained activities on an ongoing basis within FOSS projects for them to succeed.

Second, in terms of cooperation, coordination, and control, FOSS projects do not escape conflicts in technical decision-making, or in choices of who gets to work on what, or who gets to modify and update what. As FOSS projects generally lack traditional project managers, then they must become self-reliant in their ability to mitigate and resolve outstanding conflicts and disagreements. Beliefs and values that shape system design choices, as well as choices over which software tools to use, and which software artifacts to produce or use, are determined through negotiation rather than administrative assignment. Negotiation and conflict management then become part of the cost that FOSS developers must bear in order for them to have their beliefs and values fulfilled. It is also part of the cost they bear in convincing and negotiating with others often through electronic communications to adopt their beliefs and values. Time, effort, and attention spent in negotiation and conflict management are not spent building and improving source code, but they do represent an investment in building and sustaining a negotiated socio-technical network of dependencies.

Third, in terms of forming alliances and building community through participation, artifacts, and tools points to a growing dependence on other FOSS projects. The emergence of non-profit foundations that were established to protect the property rights of large multi-component FOSS projects create a demand to sustain and protect such foundations. If a foundation becomes too bureaucratic as a result to streamline its operations, then this may drive contributors away from a project. So, these foundations need to stay lean, and not become a source of occupational careers, in order to survive and evolve. Similarly, as FOSS projects give rise to new types of requirements for community building, community software, and community information sharing systems, these requirements need to be addressed and managed by FOSS project contributors in roles above and beyond those involved in enhancing the source code of a FOSS project. FOSS alliances and communities depend on a rich and growing web of socio-technical relations. Thus, if such a web begins to come apart, or if the new requirements cannot be embraced and satisfied, then the FOSS project community and its alliances will begin to come apart.

Fourth, in terms of the co-evolution of FOSS systems and community, as already noted, individual and shared resources of people's time, effort, attention, skill, sentiment (beliefs and values), and computing resources are part of the socio-technical web of FOSS. Reinventing existing software systems as FOSS coincides with the emergence or reinvention of a community who seeks to make such system reinvention occur. FOSS systems are common pool resources [Ostrom, Calvert, and Eggerston 1990] that require collective action for their development, mobilization, use, and evolution. Without the collective action of the FOSS project community, the common pool will dry up, and without the common pool, the community begins to fragment and disappear, perhaps to search for another pool elsewhere.

Last, empirical studies of FOSSD are expanding the scope of what we can observer, discover, analyze, or learn about how large software systems can be or have been developed. In addition to traditional methods used to investigate FOSSD like reflective practice, industry polls, survey research, and ethnographic studies, comparatively new techniques for mining software repositories and multi-modal modeling and analysis of the socio-technical processes and networks found in sustained FOSSD projects show that the empirical study of FOSSD is growing and expanding. This in turn will contribute to and help advance the empirical science in fields like software engineering, which previously were limited by restricted access to data characterizing large, proprietary software development projects. Thus, the future of empirical studies of software development practices, processes, and projects will increasingly be cast as studies of FOSSD efforts.

Conclusions

Free and open source software development is emerging as an alternative approach for how to develop large software systems. FOSSD employs new types and new kinds of socio-technical work practices, development processes, and community networking when compared to those found in industrial software projects, and those portrayed in software engineering textbooks [Sommerville 2004]. As a result, FOSSD offer new types and new kinds of practices, processes, and organizational forms to discover, observe, analyze, model, and simulate. Similarly, understanding how FOSSD practices, processes, and projects are similar to or different from traditional software engineering counterparts is an area ripe for further research and comparative study. Many new research opportunities exist in the empirical examination, modeling, and simulation of FOSSD activities, efforts, and communities.

FOSSD project source code, artifacts, and online repositories represent and offer new publicly available data sources of a size, diversity, and complexity not previously available for research, on a global basis. For example, software process modeling and simulation research and application has traditionally relied on an empirical basis in realworld processes for analysis and validation. However, such data has often been scarce, costly to acquire, and is often not available for sharing or independent re-analysis for reasons including confidentiality or non-disclosure agreements. FOSSD projects and project artifact repositories contain process data and product artifacts that can be collected, analyzed, shared, and be re-analyzed in a free and open source manner. FOSS poses the opportunity to favorably alter the costs and constraints of accessing, analyzing, and sharing software process and product data, metrics, and data collection instruments. FOSSD is thus poised to alter the calculus of empirical software engineering [Cook, *et al*. 1998, Harrison 2001, Scacchi 2006a]. Software process discovery, modeling, and simulation research [e.g., Jensen and Scacchi 2006a] is one arena that can take advantage of such a historically new opportunity. Another would be examining the effectiveness and efficiency of traditional face-to-face-to-artifact software engineering approaches or processes for software inspections [e.g., Ebenau and Strauss 1994, Seaman and Basili 1998] compared to the online peer reviews prevalent in FOSSD efforts.

Last, through a survey of empirical studies of FOSSD projects and other analyses presented in this article, it should be clear there are an exciting variety and diversity of opportunities for new research into software development processes, work practices, project/community dynamics, and related socio-technical interaction networks. Thus, you are encouraged to consider how your efforts to research or apply FOSSD concepts, techniques, or tools can be advanced through studies that examine FOSSD activities, artifacts, and projects.

Acknowledgments

The research described in this chapter has been supported by grants #0083075, #0205679, #0205724, #0350754, and #0534771 from the U.S. National Science Foundation. No endorsement implied. Mark Ackerman at University of Michigan, Ann Arbor; Les Gasser at University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; Margaret Elliott, Chris Jensen, and others at the UCI Institute for Software Research are collaborators on the research described here.

References

Antoniades, I.P., Samoladas, I., Stamelos, I., Angelis, L., and Bleris, G.L., (2005). Dynamic Simulation Models of the Open Source Development Process, in S. Koch (ed.), *Free/Open Source Software Development*, 174-202, Idea Group Publishing, Hershey, PA.

Benkler, Y. (2006). *The Wealth of Networks: How Social Production Transforms Markets and Freedom*, Yale University Press, New Haven, CT.

Bergquist, M. and Ljungberg, J., (2001). The power of gifts: organizing social relationships in open source communities, *Info. Systems J.*, 11, 305-320.

Beyer, H. and Holtzblatt, K., (1997). *Contextual Design: A Customer-Centered Approach to Systems Designs*, Morgan Kaufmann Publishers, San Francisco, CA.

Bonaccorsi, A. and Rossi, C., (2006). Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business. *Knowledge Technology & Policy,* 18(4), Winter*,* 40-64.

Capilupppi, A., Lago, P. and Morisio, M., (2003). Evidences in the Evolution of OS projects through Changelog Analyses, *Proc. 3rd Workshop on Open Source Software Engineering,* Portland, OR.

Chen, K., Schach, S.R., Yu, L., Offutt, J., and Heller, G. (2004). Open Source Change Logs, *Empirical Software Engineering*, 9(2), 197-210.

Ciborra, C. (2004). *The Labyrinths of Information: Challenging the Wisdom of Systems*, Oxford University Press, Oxford, UK.

Cook, J.E., Votta, L.G., and Wolf, A.L., (1998). Cost-Effective Analysis of In-Place Software Processes, *IEEE Trans. Software Engineering*, 24(8), 650-663.