

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 software¹. 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 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 Tools

My pages Projects Community

Search

Go

Advanced search

POWERED BY COLLABNET

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, framework, 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 an 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 - [perforce 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 quest 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 Pollock 1999] in the course of developing, using, and evolving FOSS systems.

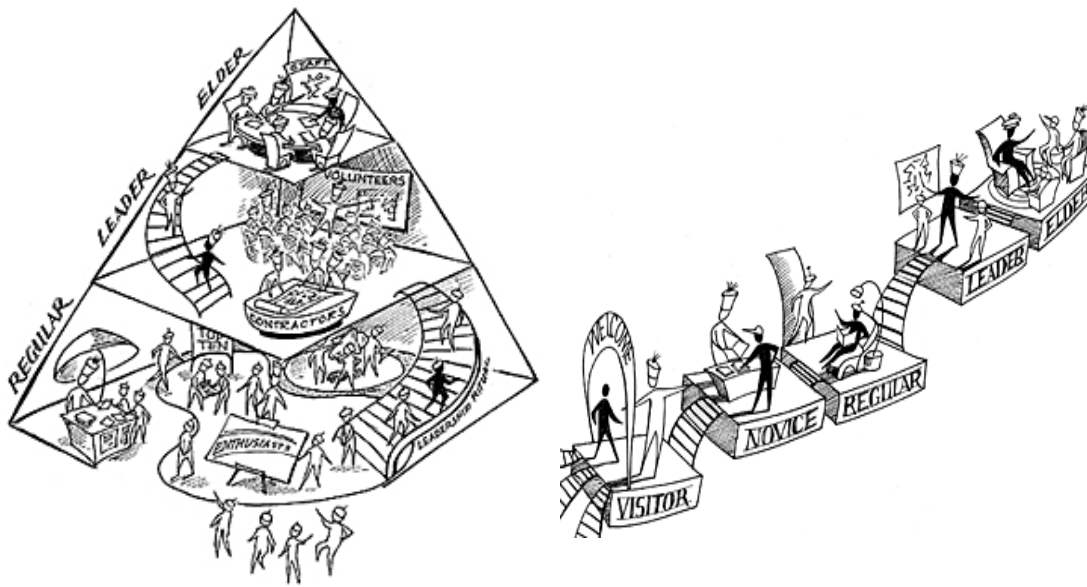


Figure 1. A visual depiction of role hierarchy within a project community (source: Kim [2000]).

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.

FOSSD informalisms

Software informalisms [Scacchi 2002] are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a FOSSD project. They are informal narrative resources that coalesce into *online document genres* (following Kwansik and Crowston 2005, Spinuzzi 2003) that are comparatively easy to use, and publicly accessible to those who want to join the project, or just browse around. Subsequently, Scacchi [2002] demonstrates how software informalisms can take the place of formalisms, like “requirement specifications” or software design notations which are seen as necessary to develop high quality software according to the software engineering community [cf. Sommerville 2004]. Yet these software informalisms often capture the detailed rationale and debates for why changes were made in particular development activities, artifacts, or source code files. Nonetheless, the contents these informalisms embody require extensive review and comprehension by a developer before contributions can be made [cf. Lanzara and Morner 2005].

The most common informalisms used in FOSSD projects include (i) communications and messages within project Email, (ii) threaded message discussion forums, bulletin boards, or group blogs, (iii) news postings, (iv) project digests, and (v) instant messaging or Internet relay chat. They also include (vi) scenarios of usage as linked Web pages, (vii) how-to guides, (viii) to-do lists, (ix) FAQs, and other itemized lists, and (x) project Wikis, as well as (xi) traditional system documentation and (xii) external publications. FOSS (xiii) project property licenses are documents that also help to define what software or related project content are protected resources that can subsequently be

shared, examined, modified, and redistributed. Finally, (xiv) open software architecture diagrams, (xv) intra-application functionality realized via scripting languages like Perl and PHP, and the ability to either (xvi) incorporate externally developed software modules or “plug-ins”, or (xvii) integrate software modules from other OSSD efforts, are all resources that are used informally, where or when needed according to the interests or actions of project participants.

All of the software informalisms are found or accessed from (xix) project related Web sites or portals. These Web environments where most FOSS software informalisms can be found, accessed, studied, modified, and redistributed [Scacchi 2002].

A Web presence helps make visible the project's information infrastructure and the array of information resources that populate it. These include FOSSD multi-project Web sites (e.g., SourceForge.net, Savannah.org, Freshment.org, Tigris.org, Apache.org, Mozilla.org), community software Web sites (PHP-Nuke.org), and project-specific Web sites (e.g., www.GNUenterprise.org), as well as (xx) embedded project source code Webs (directories), (xxi) project repositories (CVS [Fogel 1999]), and (xxii) software bug reports and (xxiii) issue tracking data base like Bugzilla (see <http://www.bugzilla.org/>).

Together, these two dozen or so types of software informalisms constitute a substantial yet continually evolving web of informal, semi-structured, or processable information resources. This web results from the hyperlinking and cross-referencing that interrelate the contents of different informalisms together. Subsequently, these FOSS informalisms are produced, used, consumed, or reused within and across FOSS development projects.

They also serve to act as both a distributed virtual repository of FOSS project assets, as well as the continually adapted distributed knowledge base through which project participants evolve what they know about the software systems they develop and use.

Competently skilled, self-organizing, and self-managed software developers

Developing complex software modules for FOSS applications requires skill and expertise in a target application domain. For example, contributing to a FOSSD project like Filezilla³ requires knowledge and skill in handling file transfer conditions, events, and protocols. Developing FOSS modules or applications in a way that enables an open architecture requires a base of prior experience in constructing open systems. The skilled use of project management tools for tracking and resolving open issues, and also for bug reports contribute to the development of such system architecture. These are among the valuable professional skills that are mobilized, brought to, or drawn to FOSS development projects [cf. Crowston 2002, 2006]. These skills are resources that FOSS developers bring to their projects.

FOSS developers organize their work as a virtual organizational form that seems to differ from what is common to in-house, centrally managed software development projects, which are commonly assumed in traditional software engineering textbooks. Within in-house development projects, software application developers and end-users often are juxtaposed in opposition to one another [cf. Curtis, Krasner and Iscoe 1988, Kling and Scacchi 1982]. Historically, Danziger [1979] referred to this concentration of software

³ See <http://filezilla.sourceforge.org>.

development skills, and the collective ability of an in-house development organization to control or mitigate the terms and conditions of system development as a "skill bureaucracy". Such a software development skill bureaucracy would seem to be mostly concerned with rule-following and rationalized decision-making, perhaps as guided by a "software development methodology" and its corresponding computer-aided software engineering tool suite.

In the decentralized virtual organization of a large ongoing FOSSD project like the Apache.org or Mozilla.org, a "skill meritocracy" [cf. Fielding 1999] appears as an alternative to the skill bureaucracy. In such a meritocracy, there is no proprietary software development methodology or tool suite in use. Similarly, there are few explicit rules about what development tasks should be performed, who should perform, when, why, or how. However, this is not to say there are no rules that serve to govern the project or collective action within it.

The rules of governance and control are informally articulated but readily recognized by project participants. These rules serve to control the rights and privileges that developers share or delegate to one another in areas such as who can commit source code to the project's shared repository for release and redistribution [cf. Fogel 1999, 2005].

Similarly, rules of control are expressed and incorporated into the open source code itself in terms of how, where, and when to access system-managed data via application program interfaces, end-user interfaces, or other features or depictions of overall system architecture. But these rules may and do get changed through ongoing project development.

Subsequently, FOSS project participants self-organize around the expertise, reputation, and accomplishments of core developers, secondary contributors, and tertiary reviewers and other peripheral volunteers [De Souza, Redmiles, and Dourish 2005, Lave and Wenger 1991]. This in turn serves to help create an easily assimilated basis for their collective action in developing FOSS [cf. Benkler 2006, Marwell and Oliver 1993, Olson 1971, Ostrom, Calvert, and Eggertsson 1993]. Thus, there is no assumption of a communal or egalitarian authority or utopian spirit. Instead what can be seen is a pragmatic, continuously negotiated order that tries to minimize the time and effort expended in mitigating decision-making conflicts while encouraging cooperation through reiterated and shared beliefs, values, norms, and other mental models [Elliott and Scacchi 2005, Espinosa, *et al.*, 2002].

Participants nearer the core have greater control and discretionary decision-making authority, compared to those further from the core [cf. Crowston and Howison 2006, De Souza, Redmiles, and Dourish 2005, Lave and Wenger 1991]. However, realizing such authority comes at the price of higher commitment of personal resources described above. Being able to make a decision stick or to convince other ongoing project participants as to the viability of a decision, advocacy position, issue or bug report, also requires time, effort, communication, and creation of project content to substantiate such an action. This authority also reflects developer experience as an interested end-user of the software modules being developed. Thus, developers possessing and exercising such skill may be intrinsically motivated to sustain the evolutionary development of their FOSS modules, so long as they are active participants in their project.

Discretionary time and effort of developers

Are FOSS developers working for "free" or for advancing their career and professional development? Following the survey results of Hars and Ou [2002] and others [FLOSS 2002, Hann, *et al.* 2002, Hertel *et al.* 2003, Lakhani *et al.* 2002, Lerner and Tirole 2000], there are many personal and professional career oriented reasons for why participants will contribute their time and effort to the sometimes difficult and demanding tasks of software development. Results from case studies in free software projects like GNUenterprise.org appear consistent with these observations [Elliott and Scacchi 2003, 2005, 2006]. These include not only self-determination, peer recognition, project affiliation or identification, and self-promotion, but also belief in the inherent value of free software [cf. DiBona *et al.* 1999, 2005, Fogel 2005, Gay 2002, Pavlicek 2000, Williams 2002].

In the practice of self-determination, no one has the administrative authority to tell a project member what to do, when, how, or why. FOSS developers can choose to work on what interests them personally. FOSS developers, in general, work on what they want, when they want. However, they remain somewhat accountable to the inquiries, reviews, and messages of others in the ongoing project, particularly with regard to software modules or functions for which they have declared their responsibility to maintain or manage as a core developer.

In the practice of peer recognition, a developer becomes recognized as an increasingly valued project contributor as a growing number of their contributions make their way into the core software modules [Benkler 2006, Bergquist and Ljungberg 2001]. In addition, nearly two-thirds of OSS developers work on 1-10 additional OSSD projects [Hars and Ou 2002, Madey, *et al.*, 2005], which also reflect a growing social network of alliances across multiple FOSS development projects [cf. Monge, *et al.* 1998, Scacchi 2005].

Project contributors who span multiple FOSS project communities serve as "social gateways" that increase the ongoing project's social mass [cf. Marwell and Oliver 1993], as well as affording opportunities for inter-project software composition and interoperation [Jensen and Scacchi 2005]. It also enables and empowers their recognition across multiple communities of FOSSD peers, which in turn reinforces their willingness to contribute their time and effort to FOSSD project communities.

In self-promotion, project participants communicate and share their experiences, perhaps from other application domains or work situations, about how to accomplish some task, or how to develop and advance through one's career. Being able to move from the project periphery towards the center or core of the development effort requires not only the time and effort of a contributor, but also the ability to communicate, learn from, and convince others as to the value or significance of the contributions [cf. Jensen and Scacchi 2006b, Lave and Wegner 1991]. This is necessary when a participant's contribution is being questioned in open project communications, not incorporated (or "committed") within a new build version, or rejected by vote of those already recognized as core developers [cf. Fielding 1999].

The last source of discretionary time and effort that has been reported is found in the freedoms and beliefs in FOSSD that are shared, reiterated and put into observable interactions. If a project participant fails to sustain or reiterate the freedoms and beliefs codified in the GPL, then it is likely the person's technical choice in the project may be called into question [Elliott and Scacchi 2003, 2005], or the person will leave the project. But understanding how these freedoms and beliefs are put into action points to another class of resources (i.e., sentimental resources) that must be mobilized and brought to bear in order to both develop FOSS systems and the global communities that surround and empower them. Social values that reinforce and sustain the ongoing project and technical norms regarding which software development tools and techniques to use (e.g., avoid the use of "non-free" software), are among the sentimental resources that are employed when participants seek to influence the choices that others in the project seek to uphold.

Trust and social accountability mechanisms

Developing complex FOSS source code and applications requires trust and accountability among project participants. Though trust and accountability in a FOSSD project may be invisible resources, ongoing software and project development work occur only when these intangible resources and mechanisms for social control are present [cf. Gallivan 2001, Hertzum, *et al.* 2002].

These intangible resources (or "social capital") arise in many forms. They include (a) assuming ownership or responsibility of a community software module, (b) voting on the approval of individual action or contribution to ongoing project software [Fielding 1999],

(c) shared peer reviewing [Benkler 2006, DiBona, *et al.* 1999, 2005], and (d) contributing gifts [Bergquist and Ljungberg 2001] that are reusable and modifiable common goods [Olson 1971, Ghosh 2005, Lessig 2005]. They also exist through the project's recognition of a core developer's status, reputation, and geek fame [Pavlicek 2000]. Without these attributions, developers may lack the credibility they need to bring conflicts over how best to proceed to some accommodating resolution. Finally, as a FOSSD project grows in terms of the number of contributing developers, end-users, and external sponsors, then project's socio-technical mass (i.e., web of interacting resources) becomes sufficient to insure that individual trust and accountability to the project are sustained and evolving [Marwell and Oliver 1993].

Thus, FOSSD efforts rely on mechanisms and conditions for gentle but sufficient social control that helps constrain the overall complexity of the project. These constraints act in lieu of an explicit administrative authority or software project management regime that would schedule, budget, staff, and control the project's development trajectory with varying degrees of administrative authority and technical competence [cf. Sommerville 2004].

Cooperation, coordination, and control in FOSS projects

Getting software developers to work together, even when they desire to cooperate is not without its challenges for coordinating and controlling who does what when, and to what they do it to. Conflicts arise in both FOSSD [Elliott and Scacchi 2003, 2005, Jensen and Scacchi 2004] and traditional software development projects [Sawyer 2001], and finding ways to resolve conflicts becomes part of the cost (in terms of social capital) that must be

incurred by FOSS developers for development progress to occur. Minimizing the occurrence, duration, and invested effort in such conflicts quickly becomes a goal for the core developers in an FOSSD project. Similarly, finding tools and project organizational forms that minimize or mitigate recurring types of conflicts also becomes a goal for experienced core developers.

Software version control tools such as the concurrent versions system, CVS--itself an FOSS system and document base [Fogel 1999]--have been widely adopted for use within FOSS projects [cf. DiBona *et al.* 1999, 2005, Feller *et al.* 2005, Fogel 2005, Pavelicek 2000]. Tools like CVS are being used as both (a) a centralized mechanism for coordinating and synchronizing FOSS development, as well as (b) an online venue for mediating control over what software enhancements, extensions, or architectural revisions will be checked-in and made available for check-out throughout the decentralized project as part of the publicly released version [cf. Ovaska, Rossi and Martiin 2003].

Software version control, as part of a software configuration management activity, is a recurring situation that requires coordination but enables stabilization and synchronization of dispersed and somewhat invisible development work [Grinter 1996]. This coordination is required due to the potential tension between centralized decision-making authority of a project's core developers and decentralized work activity of project contributors when two or more autonomously contributed software source code/content updates are made which overlap, conflict with one another, or generate unwanted side-effects [Grinter 2003]. It is also practiced as a way to manage, track, and control both

desired and undesired dependencies within the source code [De Souza *et al.* 2005], as well as among its surrounding informalisms [Scacchi 2002, 2004]. Tools like CVS thus serve to help manage or mitigate conflicts over who gets to modify what, at least as far as what changes or updates get included in the next software release from a project. However, the CVS administrator or configuration control policies provide ultimate authority and control mediated through such systems.

Each project team, or CVS repository administrator in it, must decide what can be checked in, and who will or will not be able to check-in new or modified software source code content. Sometimes these policies are made explicit through a voting scheme [Fielding 1999], or by reference to coding or data representation standards [Iannacci 2005a], while in others they are left informal, implicit, and subject to negotiation as needed. In either situation, version updates must be coordinated in order for a new system build and release to take place. Subsequently, those developers who want to submit updates to the project's shared repository rely extensively on online discussions that are supported using "lean media" such as threaded messages (via discussion forum, bulletin board, or similar) posted on a Web site [Yamauchi *et al.* 2000], rather than through onerous system configuration control boards. Thus, software version control, system build and release is a coordination and control process mediated by the joint use of versioning, system building, and communication tools [Erenkrantz 2003].

FOSSD projects teams can take the organizational form of a *layered* or *pyramid meritocracy* [cf. Fielding 1999, Kim 2000, Scacchi 2004] operating as a dynamically organized virtual enterprise [Crowston 2002, Noll 1999]. A layered meritocracy is a

hierarchical organizational form that centralizes and concentrates certain kinds of authority, trust, and respect for experience and accomplishment within the team [cf. Crowston and Howison 2006]. Such an organizational form also makes administrative governance more tractable and suitable, especially when a FOSS project seeks to legally constitute a non-profit foundation to better address its legal concerns and property rights [O'Mahony 2003]. However, it does not necessarily imply the concentration of universal authority into a single individual or directorial board, since decision-making may be shared among core developers who act as peers at the top layer, and they may be arrayed into overlapping groups with other project contributors with different responsibilities and interest areas.

As seen earlier in Figure 1, there is a layered or pyramidal form of a meritocracy common to many FOSS projects. In this form, software development work appears to be logically centralized, while being physically distributed in an autonomous and decentralized manner [Noll and Scacchi 1999]. However, it is neither simply a "cathedral" or a "bazaar", as these terms have been used to describe alternative ways of organizing FOSSD projects. Instead, when layered meritocracy operates as a virtual enterprise, it relies on *virtual project management* (VPM) to mobilize, coordinate, control, build, and assure the quality of FOSS development activities. It may invite or encourage system contributors to come forward and take a shared, individual responsibility that will serve to benefit the FOSS collective of user-developers. VPM requires multiple people to act in the roles of team leader, sub-system manager, or system module owner in a manner that may be short-term or long-term, based on their skill, accomplishments, availability and belief in ongoing project development.

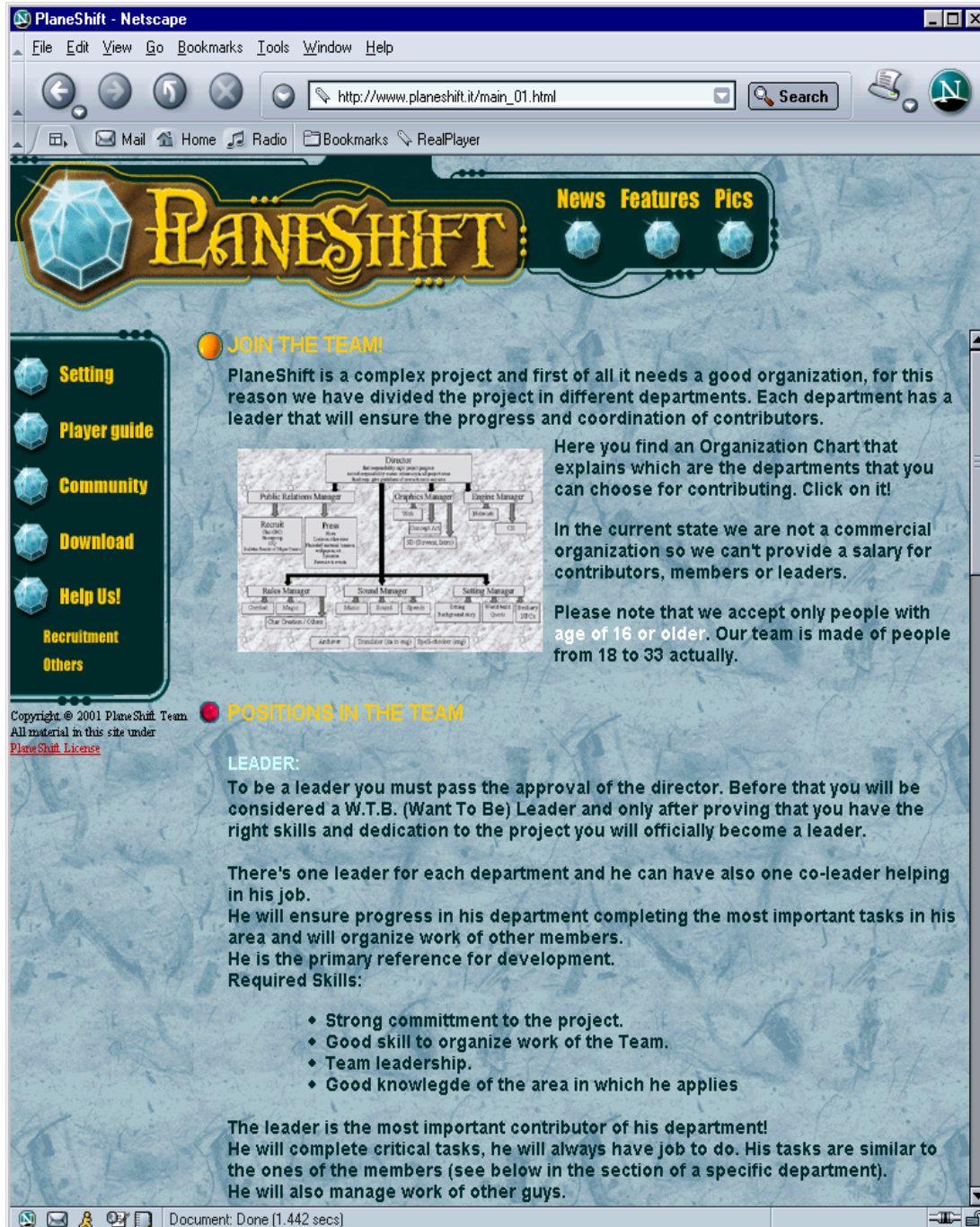


Exhibit 4. Description of virtual project management skills implied for a “Team Leader”. (source. http://www.planeshift.it/main_01.html, October 2003; also in Scacchi [2004]).

This implied requirement for virtual project management can be seen within Exhibit 4, from the FOSS project developing *Planeshift*, a free massively multiplayer online role-playing game.

Project participants higher up in the meritocracy have greater perceived authority than those lower down. But these relationships are only effective as long as everyone agrees to their makeup and legitimacy. Administrative or coordination conflicts that cannot be resolved may end up either by splitting or forking a new system version with the attendant need to henceforth take responsibility for maintaining that version [cf. Iannacii 2005a], by reducing one's stake in the ongoing project, or by simply conceding the position in conflict.

Virtual project management exists within FOSS communities to enable control via project decision-making, Web site administration, and CVS repository administration in an effective manner. Similarly, VPM exists to mobilize and sustain the use of privately owned resources (e.g., Web servers, network access, site administrator labor, skill and effort) available for shared use or collective reuse by the ongoing project.

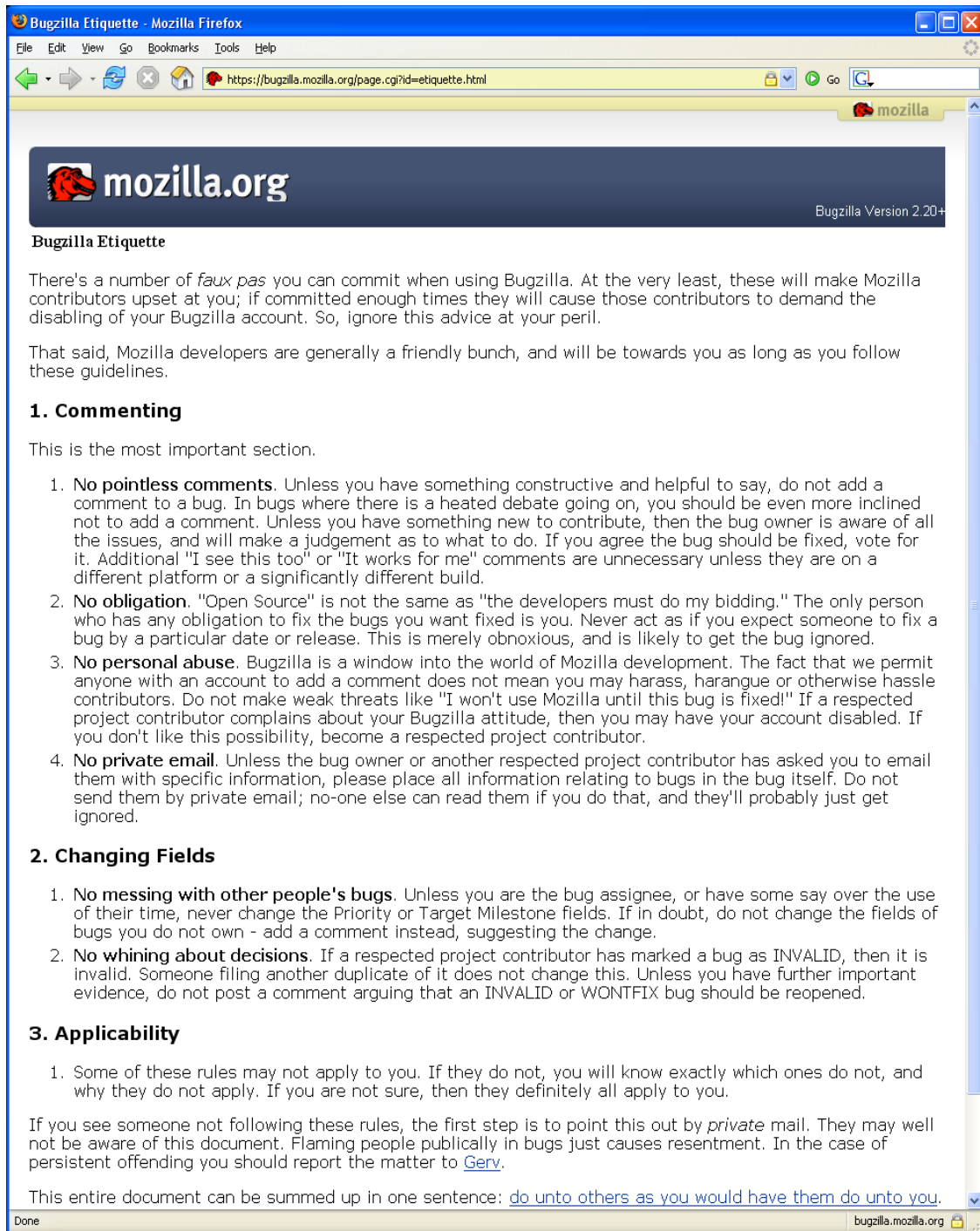
Traditional software project management stresses planning and control activities. In contrast, Lessig [2000] and others [De Souza, Redmiles, and Dourish 2005, Hakken 1999, Lanzara and Morten 2005, Scacchi 2002] observe that source code and other online artifacts are an institutional forum for collective action [O'Mahony 2003, Ostrom, Calvert, and Eggertsson 1990] that intentionally or unintentionally realizes a mode of social control on those people who develop or use it. In the case of FOSS development, Lessig's observation would suggest that the source code controls or constrains end-user and developer interaction, while the code in software development tools, Web sites, and project assets accessible for download controls, constrains, or facilitates developer

interaction with the evolving FOSS system code. CVS is a tool that enables some form of social control. However, the fact that the source code to these systems is available in a free and open source manner offers the opportunity to examine, revise, and redistribute patterns of social control and interaction in ways that favor one form of project organization, system configuration control, and user-developer interaction over others.

Many FOSSD project post guidelines for appropriate and inappropriate ways of reporting and discussing bugs, unintended features, or flaws in the current FOSS system release. These guidelines are embodied in online documents/artifacts that developers choose to follow in ways that suggest these developers have elevated informalisms into community standards that act to control appropriate behavior within FOSSD projects. Exhibit 5 provides an example of such guidelines and the rules it suggests for how to best report bugs within Mozilla projects (like the Firefox Web browser or Thunderbird email client projects) when using the Bugzilla bug reporting system.

Beyond this, the ability for the eyes of many developers to review or inspect source code, system build and preliminary test results [Porter *et al.* 1997, 2006], as well as responses to bug reports, also realizes peer review and the potential for embarrassment as a form of indirect social control over the timely actions of contributing FOSS developers [cf. Pavelicek 2000]. Thus, FOSSD allows for this dimension of VPM to be open for manipulation by the core developers, so as to encourage certain patterns of software development and social control, and to discourage others that may not advance the collective needs of FOSSD project participants. Subsequently, FOSSD projects are

managed, coordinated and controlled, though without the roles for traditional software engineering project managers [cf. Sommerville 2004].



The screenshot shows a Mozilla Firefox browser window displaying the 'Bugzilla Etiquette' page. The browser's address bar shows the URL 'https://bugzilla.mozilla.org/page.cgi?id=etiquette.html'. The page header features the Mozilla logo and the text 'Bugzilla Version 2.20+'. The main content area is titled 'Bugzilla Etiquette' and contains the following text: 'There's a number of *faux pas* you can commit when using Bugzilla. At the very least, these will make Mozilla contributors upset at you; if committed enough times they will cause those contributors to demand the disabling of your Bugzilla account. So, ignore this advice at your peril. That said, Mozilla developers are generally a friendly bunch, and will be towards you as long as you follow these guidelines.'

1. Commenting

This is the most important section.

- 1. No pointless comments.** Unless you have something constructive and helpful to say, do not add a comment to a bug. In bugs where there is a heated debate going on, you should be even more inclined not to add a comment. Unless you have something new to contribute, then the bug owner is aware of all the issues, and will make a judgement as to what to do. If you agree the bug should be fixed, vote for it. Additional "I see this too" or "It works for me" comments are unnecessary unless they are on a different platform or a significantly different build.
- 2. No obligation.** "Open Source" is not the same as "the developers must do my bidding." The only person who has any obligation to fix the bugs you want fixed is you. Never act as if you expect someone to fix a bug by a particular date or release. This is merely obnoxious, and is likely to get the bug ignored.
- 3. No personal abuse.** Bugzilla is a window into the world of Mozilla development. The fact that we permit anyone with an account to add a comment does not mean you may harass, harangue or otherwise hassle contributors. Do not make weak threats like "I won't use Mozilla until this bug is fixed!" If a respected project contributor complains about your Bugzilla attitude, then you may have your account disabled. If you don't like this possibility, become a respected project contributor.
- 4. No private email.** Unless the bug owner or another respected project contributor has asked you to email them with specific information, please place all information relating to bugs in the bug itself. Do not send them by private email; no-one else can read them if you do that, and they'll probably just get ignored.

2. Changing Fields

- 1. No messing with other people's bugs.** Unless you are the bug assignee, or have some say over the use of their time, never change the Priority or Target Milestone fields. If in doubt, do not change the fields of bugs you do not own - add a comment instead, suggesting the change.
- 2. No whining about decisions.** If a respected project contributor has marked a bug as INVALID, then it is invalid. Someone filing another duplicate of it does not change this. Unless you have further important evidence, do not post a comment arguing that an INVALID or WONTFIX bug should be reopened.

3. Applicability

1. Some of these rules may not apply to you. If they do not, you will know exactly which ones do not, and why they do not apply. If you are not sure, then they definitely all apply to you.

If you see someone not following these rules, the first step is to point this out by *private* mail. They may well not be aware of this document. Flaming people publically in bugs just causes resentment. In the case of persistent offending you should report the matter to [Gerv](#).

This entire document can be summed up in one sentence: [do unto others as you would have them do unto you.](#)

The browser's status bar at the bottom shows 'Done' and the address 'bugzilla.mozilla.org'.

Exhibit 5. Guidelines for appropriate behavior when reporting bugs in Mozilla.org FOSS projects when using the Bugzilla bug reporting system.
(source: <https://bugzilla.mozilla.org/page.cgi?id=etiquette.html>, June 2006).

Alliance formation, inter-project social networking and community development

How does the gathering of FOSS developers give rise to a more persistent self-sustaining organization or project community? Through choices that developers make for their participation and contribution to a FOSSD project, they find that there are like-minded individuals who also choose to participate and contribute to a project. These software developers find and connect with each other through FOSSD Web sites and online discourse (e.g., threaded discussions on bulletin boards) [Monge *et al.* 1998], and they find they share many technical competencies, values, and beliefs in common [Crowston and Scozzi 2002, Espinosa *et al.* 2002, Elliott and Scacchi 2005]. This manifests itself in the emergence of an alliance of FOSSD projects that share either common interests or development methods, like those for “open source software engineering” identified in the left column in Exhibit 1, in external projects that adopt a given FOSS system (e.g., OGRE) as the core system for subsequent application development as seen in Exhibit 6, or in a occupational network of FOSS developers [Elliott and Scacchi 2005].

Becoming a central 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. 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.* 2002, 2005, Scacchi 2005].

Featured Projects

32 sub-albums and no images in this album on 2 pages

Gallery: **OGRE Gallery** ↗

▷ 1 ◀ 2 ◦

▷▷ ▷|

A selection of screenshots from real-world projects which are using OGRE today.



Album: **Agent Hugo**

Changed:
07/09/06
Contains: 8
items.
Viewed: 1083
times.



Album: **The Blob**

Changed:
06/30/06
Contains: 16
items.
Viewed: 2458
times.



Album: **Freaks**

Changed:
04/25/06
Contains: 4
items.
Viewed: 13342
times.



Album: **Legend Of The Dragon**

Changed:
04/24/06
Contains: 7
items.



Album: **oFusion**

Changed:
04/03/06
Contains: 8
items.



Album: **Sector 13**

Changed:
04/03/06
Contains: 10
items.

Exhibit 6. A partial view of an alliance of external FOSS game development projects that use the OGRE system (cf. Exhibit 2). (source: http://www.ogre3d.org/index.php?set_albumName=album07&option=com_gallery&Itemid=55&include=view_album.php, June 2006)

“Linchpin developers” [Madey *et al.* 2005] participate in or span multiple FOSSD projects. In so doing, they create alliances between otherwise independent FOSSD projects. Figure 2 depicts an example of a social network that clusters 24 FOSS developers within 5 FOSSD projects interconnected through two linchpin developers [Madey *et al.* 2005].

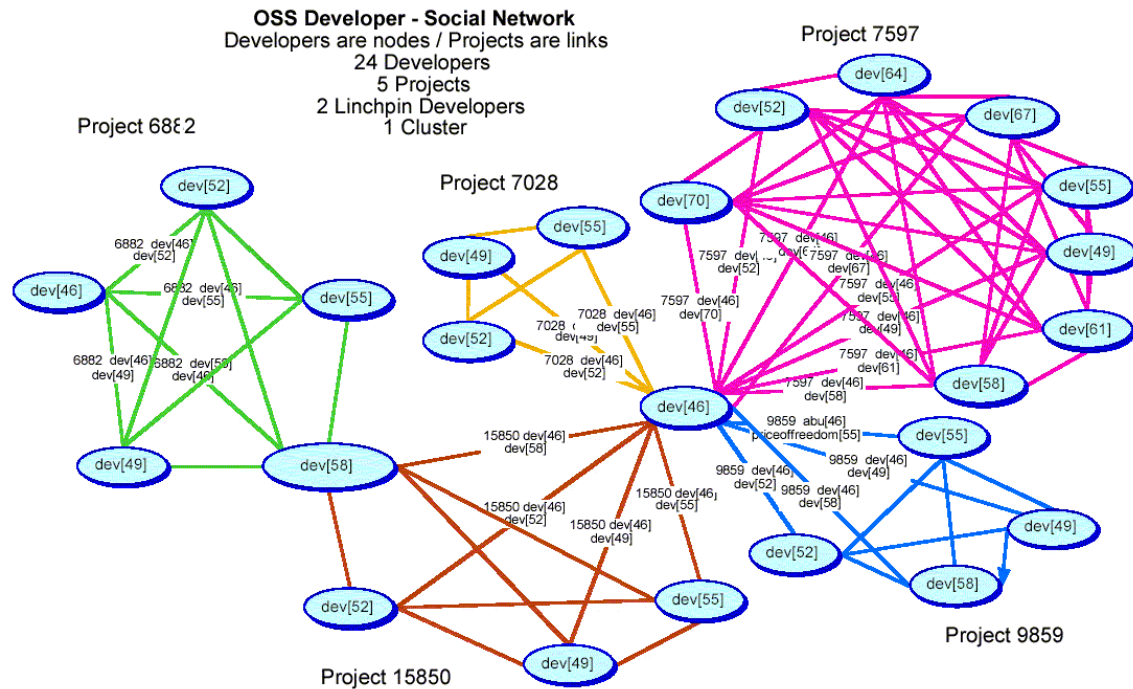


Figure 2. A social network that clusters 24 developers in five FOSS projects through two key developers into a larger project community [source: Madey *et al.* 2002].

Multi-project clustering and interconnection enables small FOSS projects to come together as a larger social network with the critical mass [Marwell and Oliver 1993] needed for their independent systems to be merged and experience more growth in size, functionality, and user base. It also enables shared architectural dependencies to arise (perhaps unintentionally) in the software components or sub-systems that are used/reused across projects [cf. DeSouza, Redmiles, and Dourish 2005, Iannacci 2005a, Ovaska, Rossi and Martiin 2003]. FOSSD Web sites also serve as hubs that centralize attention for what is happening with the development of the focal FOSS system, its status, participants and contributors, discourse on pending/future needs, etc.

Sharing beliefs, values, communications, artifacts and tools among FOSS developers enables not only cooperation, but also provides a basis for shared experience, camaraderie, and learning [cf. Espinosa *et al* 2002, Fischer 2001, Huntley 2003, Lave and Wenger 1991]. FOSS developers participate and contribute by choice, rather than by assignment, since they find that conventional software development work provides the experience of working with others who are assigned to a development effort, whether or not they find that share technical approaches, skills, competencies, beliefs or values. As a result, FOSS developers find they get to work with people that share their many values and beliefs in common, at least as far as software development. Further, the values and beliefs associated with free software or open source software are both signaled and institutionalized in the choice of intellectual property licenses (e.g., GPL) that FOSSD projects adopt and advocate. These licenses in turn help establish norms for developing free software or open source software, as well as for an alliance with other FOSSD projects that use the same licenses.

Almost half of the over 120K FOSS projects registered at SourceForce.net Web portal (as of July 2006-see Exhibit 7 later) employ the GNU General Public License (GPL) for free (as in freedom) software. The GPL seeks to preserve and reiterate the beliefs and practices of sharing, examining, modifying and redistributing FOSS systems and assets as common property rights for collective freedom [Gay 2002, Lessig 2005, Williams 2002]. A few large FOSSD project that seek to further protect the collective free/open intellectual property rights do so through the formation of legally constituted non-profit organizations or foundations (e.g., Free Software Foundation, Apache Software Foundation, GNOME Foundation) [O'Mahony 2003]. Other OSS projects, because of the

co-mingling of assets that were not created as free property, have adopted variants that relax or strengthen the rights and conditions laid out in the GPL. Dozens of these licenses now exist, with new ones continuing to appear (cf. www.opensource.org). Finally, when OSSD projects seek to engage or receive corporate sponsorship, and the possible co-mingling of corporate/proprietary intellectual property, then some variation of a non-GPL open source license is employed, as a way to signal a “business friendly” OSSD project, and thus to encourage participation by developers who want to work in such a business friendly and career enhancing project [Hann *et al.* 2002, Sharma *et al.* 2002, West and O'Mahony 2005].

Community development and system development

Developing FOSS systems is a project team building process that must be institutionalized within a community [Sharma *et al.* 2002, Smith and Pollock 1999, Preece 2000, Ye *et al.* 2005] for its software informalisms (artifacts) and tools to flourish. Downloading, installing, and using FOSS systems acquired from other FOSS Web sites is also part of a community building process [Kim 2000], while Exhibit 6 reiterates that many external game development project use the OGRE free software. Adoption and use of FOSS project Web sites are a community wide practice for how to publicize and share FOSS project assets. These Web sites can be built using FOSS Web site content management systems (e.g., PHP-Nuke) to host project contents that can be served using FOSS Web servers (Apache), database systems (MySQL) or application servers (JBoss), and increasingly accessed via FOSS Web browsers (Mozilla). Furthermore, ongoing FOSS projects may employ dozens of FOSS development tools, whether as standalone systems like the software version control system CVS, as

integrated development environments like NetBeans or Eclipse, or as sub-system components of their own FOSS application in development. These projects similarly employ asynchronous systems for project communications that are persistent, searchable, traceable, public and globally accessible [Yamauchi *et al.* 2000].

FOSS systems, hyperlinked artifacts and tools, and project Web sites serve as venues for socializing, building relationships and trust, sharing and learning with others. “Linchpin developers” [Madey *et al.* 2005] act as community forming hubs that enable independent small FOSS projects to come together as a larger social network with the critical mass [Marwell and Oliver 1993] needed for their independent systems to be merged and experience more growth in size, functionality, and user base. Whether this trend is found in traditional or closed source software projects is unclear. Multi-project FOSS Web sites (e.g., Tigris.org in Exhibit 1 or SourceForge.org in Exhibit 7) also serve as hubs or “community cores” that centralize attention for what is happening with the development of focal FOSS systems, their status, participants and contributors, discourse on pending/future needs, etc. Furthermore, by their very nature, these Web sites are generally global in reach and publicly accessible. This means the potential exists for contributors to come from multiple remote sites (geographic dispersion) at different times (24/7), from multiple nations, representing the interests of multiple cultures or ethnicity. Thus, multi-project FOSS Web sites help to make visible online virtual organizations, inter-project alliances, community and social networks that can share resources, artifacts, interests, and source code [cf. Fischer 2001].

All of these conditions point to *new kinds of requirements for software development projects*—for example, community building requirements, community software requirements, and community information sharing system (Web site and interlinked communication channels for email, forums, and chat) requirements [Scacchi 2002, Truex, Baskerville and Klien 1999]. These requirements may entail both functional and non-functional requirements, but they will most typically be expressed using FOSS informalisms, rather than using formal notations based on some system of mathematical logic known by few.

Community building, alliance forming, and participatory contributing are essential and recurring activities that enable FOSSD projects to persist without central corporate authority. Thus, linking people, systems, and projects together through shared artifacts and sustained online discourse enables a sustained social network [Lopez-Fernandez *et al.* 2006, Madey *et al.* ,2002, 2005] and socio-technical community, Web-based information infrastructure [Jensen and Scacchi 2005], and network of alliances [Iannacci 2005b, Monge *et al.* 1998] to emerge.

Therefore interesting problems arise when investigating how best to model or simulate the FOSSD processes that facilitate and constrain the co-development and co-evolution of FOSS project communities and the software systems they produce. The point is not to separate the development and evolution processes of the software system from its community, since each is co-dependent on the other, and the success of one depends on the success of the other. Thus, it appears that FOSSD processes and practices should be modeled and simulated as integrating and intertwining processes.

FOSS as a multi-project software ecosystem

As noted above, many FOSSD projects have become interdependent through the networking of software developers, development artifacts, common tools, shared Web sites, and computer-mediated communications. What emerges from this is a kind of *multi-project software ecosystem*, whereby ongoing development and evolution of one FOSS system gives rise to propagated effects, architectural dependencies, or vulnerabilities in one or more of the projects linked to it [Jensen and Scacchi 2005]. For example, Figure 3 depicts a software ecosystem primarily consisting of FOSS projects (each project denoted by a cloud-like shape, and the interrelationship of these project clouds denoting the ecosystem).

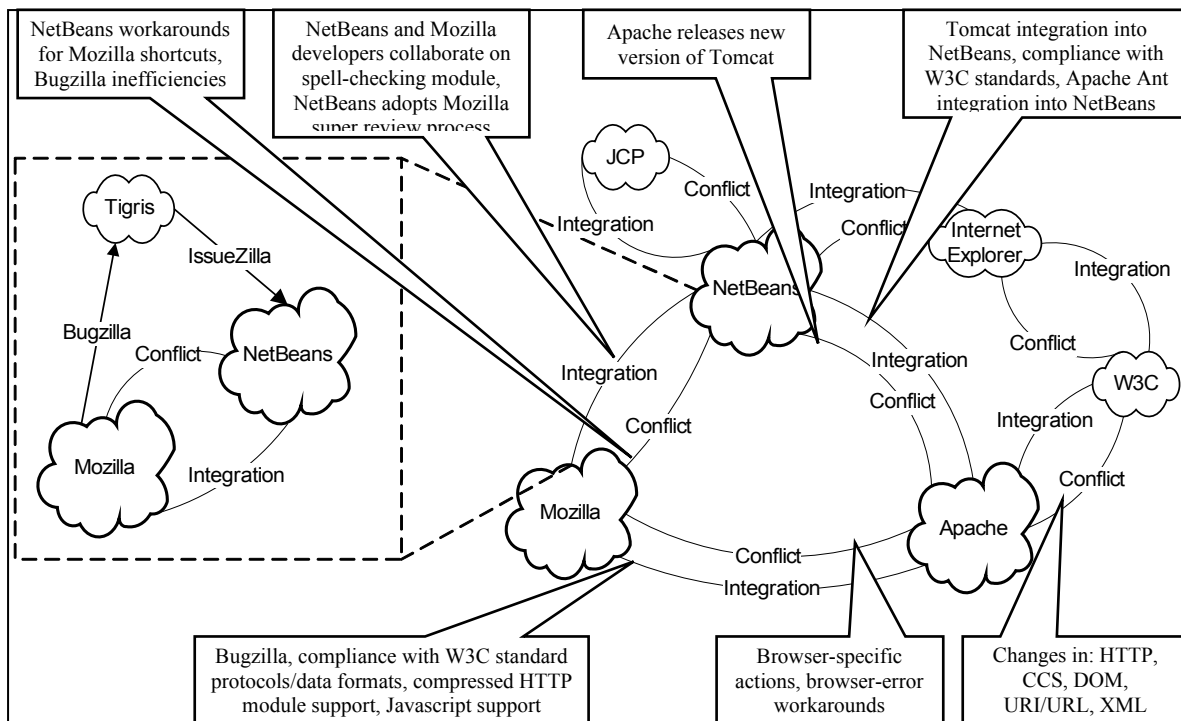


Figure 3: Visualizing cooperative integrations and conflicts among an ecosystem of interrelated FOSS projects (source: Jensen and Scacchi 2005).

This particular software ecosystem highlights relationships between three large FOSS projects, the Mozilla.org Web Browser, the Apache.org Web server, and the

NetBeans.org interactive development environment for Web-based Java applications. It also collectively forms a central part of the software infrastructure for the Web⁴, along with other FOSS projects that support each of these three focal projects. It further highlights examples of integration and conflict issues that have emerged as these three core Web systems as each has evolved on its own, as well as co-evolved with the others. Details on the integration and conflict issues are further described in Jensen and Scacchi [2005].

Interdependencies are most apparent when FOSSD project share source code modules, components, or sub-systems. In such situations, the volume of source code of an individual FOSSD project may appear to grow at a super-linear or exponential rate [Scacchi 2006a, Schach *et al.* 2002, Smith *et al.* 2004] when modules, components, or sub-systems are integrated in whole into an existing FOSS system [Scacchi 2006a]. Such an outcome, which economists and political scientists refer to as a “network externality” [Ostrom, Calvert and Eggertssons 1990], may be due to the import or integration of shared components, or the replication and tailoring of device, platform, or internationalization specific code modules. Such system growth patterns therefore seem to challenge the well-established laws of software evolution [Lehman 1980, 2002]. Thus, software evolution in a multi-project FOSS ecosystem is a process of co-evolution of interrelated and interdependent FOSSD projects, people, artifacts, tools, code, and project-specific processes.

⁴ Figure 3 also indicates the non-FOSS like Microsoft’s Internet Explorer Web browser is part of the software ecosystem for the Web software infrastructure. The Java Community Process (JCP) and World Wide Web Committee (W3C) respectively denote a software application coding compatibility assessment process, and a committee of diverse parties who collectively act to define Web standards for markup languages (HTML) and data communication protocols (http), which are central to the interoperation of Web browsers, Web servers, and Web applications.

It seems reasonable to observe that the world FOSSD is not the only place where multi-project software ecosystems emerge, as software sharing or reuse within traditional software development enterprises is common. However, the process of the co-evolution of software ecosystems found in either traditional or FOSSD projects is mostly unknown. Thus, co-evolution of interdependent software systems and standards for interoperability within an FOSS ecosystem represents an opportunity for research that investigates understanding such a software evolution process through studies supported by modeling and simulation techniques [e.g., Antoniadou, *et al.* 2005, Smith *et al.* 2004].

Co-evolving socio-technical systems for FOSS

Software maintenance, in the form of the addition/subtraction of system functionality, debugging, restructuring, tuning, conversion (e.g., internationalization), and migration across platforms, is a widespread, recurring process in FOSS development communities. Perhaps this is not surprising since maintenance is generally viewed as *the* major cost activity associated with a software system across its life cycle [cf. Sommerville 2004]. However, this traditional characterization of software maintenance does not do justice for what can be observed to occur within different FOSS communities. Instead, it may be better to characterize a key evolutionary dynamic of FOSS as *reinvention* [cf. Scacchi 2004]. Reinvention is enabled through the sharing, examination, modification, and redistribution of concepts and techniques that have appeared in closed source systems, research and textbook publications, conferences, and the interaction and discourse between developers and users across multiple FOSS projects. Thus, reinvention is a continually emerging source of improvement in FOSS functionality and quality, as well

as also a collective approach to organizational learning in FOSS projects [Fischer 2001, Huntley 2003, Lave and Wenger 1991].

Many of the largest and most popular FOSS systems like the Linux Kernel [Schach *et al.* 2002], GNU/Linux distributions [Iannacci 2005a, O'Mahony 2003], GNOME user interface [German 2003] and others are growing at an exponential rate, as is their internal architectural complexity [Schach *et al.* 2002]. On the other hand the vast majority of FOSS projects are small, short-lived, exhibit little/no growth, and often only involve the effort of one developer [Capiluppi *et al.* 2003, Madey *et al.* 2005]. In this way, the overall trend derived from samples of 400-40K FOSS projects registered at the SourceForge.net Web portal reveals a power law distribution common to large self-organizing systems. This means a few large projects have a critical mass of at least 5-15 core FOSS developers [Mockus, Fielding and Herbsleb 2002] that act in or share project leadership roles [Fielding 1999] that are surrounded by dozens to hundreds of other contributors in secondary or tertiary roles, and hundreds to millions of end users in the distant periphery. The FOSS projects that attain and sustain such critical mass are those that inevitably garner the most attention, software downloads, and usage. On the other hand, the vast majority of FOSS projects are small, lacking in critical mass, and thus unlikely to thrive and grow.

The layered meritocracies that arise in FOSS projects tend to embrace incremental innovations such as evolutionary mutations to an existing software code base over radical innovations. Radical change involves the exploration or adoption of untried or sufficiently different system functionality, architecture, or development methods. Radical

software system changes might be advocated by a minority of code contributors who challenge the status quo of the core developers. However, their success in such advocacy usually implies creating and maintaining a separate version of the system, and the potential loss of a critical mass of other FOSS developers. Thus, incremental mutations tend to win out over time [cf. Scacchi 2004].

FOSS systems seem to evolve through minor improvements or mutations that are expressed, recombined, and redistributed across many releases with short duration life cycles. End-users of FOSS systems who act as contributing developers or maintainers continually produce these mutations. These mutations appear to coalesce in daily system builds. These modifications or updates are then expressed as a tentative alpha, beta, release candidate, or stable release versions that may survive redistribution and review, then subsequently be recombined and re-expressed with other new mutations in producing a new stable release version. As a result, these mutations articulate and adapt an FOSS system to what its developer-users want it to do in the course of evolving and continually reinventing the system.

Last, closed source software systems that were thought to be dead or beyond their useful product life or maintenance period may be *revitalized* through the redistribution and opening of their source code. However, this may only succeed in application domains where there is a devoted collective of enthusiastic user-developers who are willing to invest their time and skill to keep the cultural heritage of their former experience with such systems alive. Scacchi [2004] provides an example for vintage arcade games now

numbering in the thousands that are being revitalized and evolved using the FOSS-based Multi-Arcade Machine Emulator (MAME).

Overall, FOSS systems co-evolve with their development communities. This means the evolution of one depends on the evolution of the other. Said differently, a FOSS project with a small number of developers (most typically one) will not produce and sustain a viable system unless/until the team reaches a larger critical mass of 5-15 core developers. However, if and when critical mass is achieved, then it may be possible for the FOSS system to grow in size and complexity at a sustained exponential rate, defying the laws of software evolution that have held for decades [Lehman 1980, 2002, Scacchi 2006a]. Furthermore, user-developer communities co-evolve with their systems in a mutually dependent manner [Elliott and Scacchi 2005, Nakakoji *et al.* 2002, O'Mahony 2003, Scacchi 2002], and system architectures and functionality grow in discontinuous jumps as independent FOSS projects decide to join forces [e.g., Nakakoji *et al* 2002, Scacchi 2006]. Whether this trend is found in traditional or closed source software projects is unclear. But what these findings and trends do indicate is that it appears that the practice of FOSS development processes is different from the processes traditionally advocated for software engineering.

FOSS as a Social Movement

Social movements reflect sustained and recurring large-scale collective activities within a society. Social movements can be characterized by (a) their recurring structural forms (e.g., boundaries around movement sub-segments, multiple centers of activity, and social

networks that link the segments and centers) and venues for action, (b) ideological beliefs, and (c) organizations whose purpose is to advance and mobilize broader interest in the movement [Snow *et al* 2004]. The OSS movement arose in the 1990's [DiBona, *et al.*, 1999, Ljungberg 2000, Scacchi 2006b, West and Dedrick 2006] from the smaller, more fervent “free software” movement [Gay 2002] started in the mid 1980's.

The OSS movement is populated with thousands of OSS development projects, each with its own Web site. Whether the OSS movement is just another computerization movement [cf. Iacono and Kling 2001], or is better recognized as a counter-movement to the proprietary or closed source world of commercial software development is unclear. For example, executives from proprietary software firms have asserted that (a) OSS is a national security threat to the U.S. [O'Dowd 2004], or (b) that OSS (specifically that covered by the GNU Public License or “GPL”) is a cancer that attaches itself to intellectual property [Greene 2001]. However, other business sources seem to clearly disagree with such characterizations and see OSS as an area for strategic investment [Gomes 2001, OSBC 2006], and there is growing support for recognizing that FOSS has become a matter of national security in the U.S. Department of Defense [MITRE 2003, Payton *et al.* 2006]. Nonetheless, more than 120K projects are registered at OSS portals like SourceForge.org, as seen in Exhibit 7, while other OSS portals like Freshment.org, and Tigris.org contain thousands more.

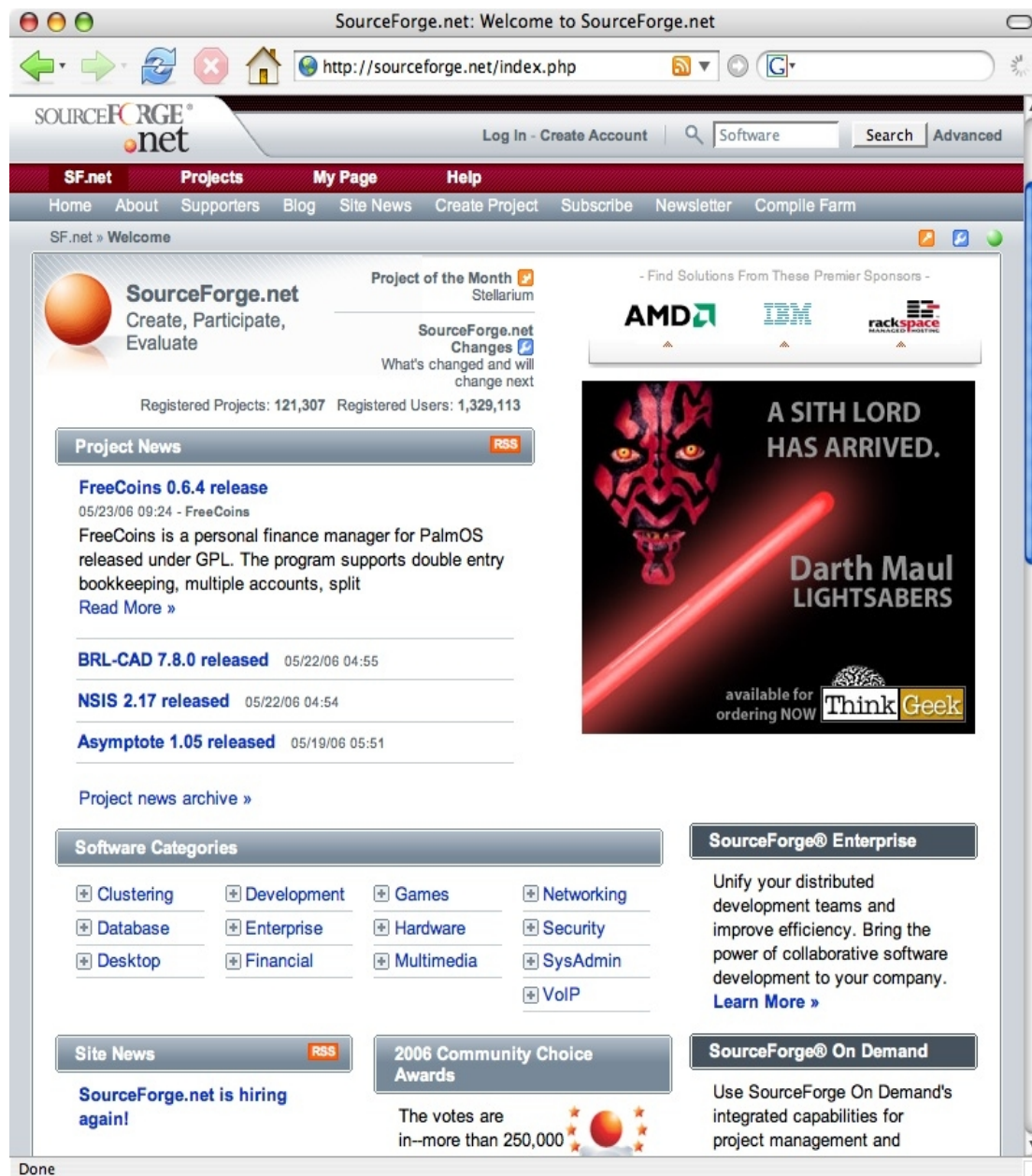


Exhibit 7. Home page of the SourceForge.net OSS Web portal, indicating more than 120K registered projects, and more than 1.3M registered user (source: <http://sourceforge.net/>, visited 7 June 2006).

The vast majority of the OSS projects at SourceForge appear to be inactive, with less than two contributing developers, as well as no software available for download, evaluation, or enhancement. However, at least a few thousand OSS projects seem to garner most of the attention and community participation, but no one project defines or leads the OSS

movement. The Linux Kernel project is perhaps the most widely known OSS project, with its celebrity leaders, like Linus Torvalds. Ironically, it is also the most studied OSS project. However, there is no basis to indicate that how things work in this project prescribe or predict what might be found in other successful OSS projects. Thus, the OSS movement is segmented about the boundaries of each OSS project, though some of the larger project communities have emerged as a result of smaller OSS projects coming together. Finally, a small set of studies [cf. Hars and Ou 2002, Koch 2005] indicate that upwards of 2/3 OSS developers contributes to two or more OSS projects, and perhaps as many as 5% contribute to 10 or more OSS projects. The density and interconnectedness of this social networking characterizes the membership and in-breeding of the OSS movement, but at the same time, the multiplicity of projects reflects its segmentation.

According to advocates [Gay 2002], Richard M. Stallman initiated the free software movement [Elliott 2006, Elliott and Scacchi 2006]. Its participants or advocates identify their affiliation and commitment by openly developing and sharing their software following the digital civil liberties expressed in the GPL. The GPL is a license agreement that promotes and protects software source code using the GPL copyright to always be available (always assuring a “copy left”), and that the code is open for study, modification, and redistribution, with these rights preserved indefinitely. Furthermore, any software system that incorporates or integrates free software covered by the GPL, is asserted henceforth to also be treated as free software. This so-called “viral” nature of the GPL is seen by some to be an “anti-business” position, which is the most commonly cited reason for why other projects have since chose to identify them as open source software [Fink 2003]. However, new/pre-existing software that does not integrate GPL source code is not infected by the GPL, even if

both kinds of software co-exist on the same computer or operating system, or that access one another through open or standards-based application program interfaces.

Surveys of OSS projects reveal that about 50% or more of all OSS projects (including the Linux Kernel project) employ the GPL [FLOSS 2002], even though there are only a few thousand of self-declared free software projects. Large OSS projects, such as the Apache Web server, KDE user interface package, Mozilla/Firefox Web browser, have chosen to not use the GPL, but to use a less restrictive, open source license. As before, free software is always open source, but open source software is not always free software. So the free software movement has emerged on its own, but increasingly it has effectively become subsumed as a segment within the larger, faster growing and faster spreading OSS movement. Subsequently, OSS licenses have become the hallmark carrier of the ideological beliefs that helps distinguish members of the free software movement, from those who share free software beliefs but prefer to be seen as open source or business-friendly developers (e.g., the Linux Kernel project). Furthermore, the use of non-GPL OSS licenses by corporate-sponsored projects [cf. O'Mahony 2003] also distinguishes those who identify themselves as OSS developers, but not practitioners or affiliates of the free software movement.

A variety of organizations, enterprises, and foundations participate in encouraging the advancement and success of OSS [Weber 2004]. Non-profit foundations have become one of the most prominent organizational forms founded to protect the common property rights of OSS projects. The Open Source Initiative (www.opensource.org) is one such foundation that

seeks to maintain the definition of what “open source software” is, and what software licenses satisfy such a definition. OSI presents its definition of OSS in a manner that is considered business friendly [Fink 2003], as opposed to “free software” which is cast by its advocates as a social movement that expresses civil liberties through software (e.g., source code as a form of free speech) [Gay 2002]. The OSI’s Bruce Perens who advocates that OSS is a viable economic and innovative alternative to proprietary software, often is juxtaposed or compared to Richard M. Stallman, who seeks to “put back the *free* in free enterprise” [Gay 2002]. Beyond this, a sign of success of the largest OSS projects is the establishment of a non-profit foundation or a not-for-profit consortium that serve as the organizational locus and legal entity that can engage in contracts and intellectual property rights agreements that benefit the project. A small but growing number of corporations in the IT, Financial Services, and other industries have taken on sponsorship of OSS projects, either as an external competitive strategy (e.g., IBM’s Eclipse project and SUN’s NetBeans project compete against Microsoft .NET products) or internal cost-reduction strategy [West and O’Mahony 2005].

Overall, recognizing that free software and OSS have facilitated the emergence of global-scale social (or computerization) movements, indicates that FOSS is increasingly permeating society at an industrial, governmental, and international level, and is doing so in ways that no prior software technology or development method has come close to achieving. Why this has come about, what consequences it portends for the future of FOSS, and whether corporate or public (government) policy initiatives will increasingly address the development, adoption, deployment, usage, and support of FOSS applications and projects, all require further study.

But it is also clear that it is increasingly unlikely that any company, government, or nation can successfully inhibit the near-term and mid-term societal dispersion of FOSS or the FOSS movements.

Research Methods for Studying FOSS

Based on the survey of studies and results emerging from empirical studies of FOSSD projects, it becomes clear that there are many promising opportunities in studying, modeling, analyzing, and comparing FOSS development processes, work practices, and community dynamics, as well as project development artifacts and source code. New sources of data associated with FOSSD participants, artifacts, tools used, and development processes are available, and new systematic samples of FOSSD projects can be articulated. Empirical studies of FOSSD can therefore be examined of the research methods employed, and that is the purpose of the following section of this chapter.

In this chapter, different studies of FOSS development were organized and characterized according to subjects grouped into different level of analysis. Subsequently, this raises questions about what kinds of research methods have been used in these studies, or might be used in future studies of FOSS. To answer such questions, it is necessary and beneficial to review what kinds of research methods and strategies have appeared in FOSS studies, in order to identify possible categories of FOSS research methods that can be practiced by or taught to future FOSS scholars. The purpose is not to endure a treatise on how to do research or how to conduct an empirical study of FOSS, but instead to highlight which studies of FOSSD used what research methods to investigate issues at one of more levels of analysis.

As research studies of FOSS can be organized in many ways, level of analysis can be construed as a constructive element when articulating a research method. A given study may explore a single or multiple levels of analysis by research study design. Other elements include the unit of analysis, terms of analysis, and mode of analysis. The *unit of analysis* focuses on what or who is being studied, across some spatio-temporal extent within some work setting. Common foci include FOSS *developer motivations*, *project teams* or *workgroup* effort, *source code*, development or communication *artifacts*, or *development processes* enacted within a project's Web Site(s) across some period of time. The choice of the unit of analysis often determines or reflects the researcher's choice for the level, terms, and mode of analysis. The *terms of analysis* refer to the choice of analytical variables and rhetorical framings that are employed to identify and describe salient features or aspects of the unit of analysis. When focusing on FOSS development processes, for example, conceptual variables like *process structure* or *process control flow* may be used to associate the partially ordered sequence of *workflow activities*, performed by participants acting in different *roles*, using *tools* to perform different activities that *access and update shared resources or artifacts*, may be used to describe observed or discovered FOSS processes. The *mode of analysis* identifies what kind of qualitative, quantitative, or triangulated schemes are employed to collect and analyze data associated with the unit of analysis.

Common FOSS research data collection and analysis modes include reflective practice and industry polls, surveys, ethnographic study, mining FOSS artifact repositories, and multi-modal modeling. As mode of analysis is core to research method, that becomes the

focus here. However, as will become clear, different research methods involve trade-offs when compared to one another, so that no single research method will be best in all situations or studies.

Reflective practice and industry poll methods

FOSS research studies often focus on the interests, motivations, perceptions, and experiences of developers or end-user organizations. Typically, the unit of analysis is the *individual agent* (most commonly a person, unitary group, or firm, but sometimes a software system, tool, or artifact type) acting within a larger actor group or community. Individual behavior, personal choices, or insider views might best be analyzed, categorized, and explained in terms of volunteered or elicited statements of their interests, motivations, perceptions, or experiences. Most of the popular treatments of OSS development [e.g., DiBona, *et al.* 1999, 2005, Fogel 1999, Pavlivcek 1999] and Free software development [e.g., Fogel 2005, Gay 2002, Williams 2002], provide insight and guidance for how FOSS development occurs, based on the first-hand experiences of those authors.

Other authors informed by such practitioner studies and informal industry/government polls (like those reported in *CIO Magazine*, MITRE [2003], OSBC [2006], Wheeler [2005], and elsewhere) seek to condense and package the experience and wisdom of these FOSS practices into practical advice that may be offered to business executives considering when and why to adopt FOSS options [e.g., Fink 2003, Goldman and Gabriel 2005].

As a FOSS research method, reflective practice often tends to (a) be uncritical with respect to prior scholarship or theoretical interpretation, or (b) employ unsystematic collection of data to substantiate pithy anecdotes or proffered conclusions. Thus, by themselves such studies offer a limited basis for further research or comparative analysis. Nonetheless, they can (and often do) offer keen insights and experience reports that may help sensitize future FOSS researchers to interesting starting points or problems to further explore.

Survey research methods

A focus on perceptions or motivations of individual participants also suggests possible attention to cognitive dimensions of FOSS development or end-user adoption. Here the quantitative survey studies of Bonaccorsi and Rossi [2006], FLOSS [2002, Ghosh and Prakash 2000], Hars and Ou [2002], Hertel, *et al.* [2003], and Lakhani, *et al.* [2002], for example, have been key in providing broad international coverage (and descriptive statistics) of why software developers of different ages, skill bases, employment status in different countries seek to join, participate in, and help sustain FOSS development projects and their surrounding communities.

The survey research studies cited above (a) critically reflect on the data and offer alternative explanations relative to established scholarship, and (b) rely on reasonably articulated questionnaire design, survey samples, and statistical analysis to plausibly substantiate their findings and conclusions. However, these surveys typically involve hundreds of individual respondents, and thus require a significant commitment of research staff expertise, time, effort, and budget to administer the survey and process the

data in the study. Furthermore, most such surveys are standalone studies, though Bonaccorsi and Rossi are one of the first to incorporate a comparative analysis of prior survey studies of motivations of FOSS developers and end-user firms who elect to join FOSS projects, while the Ghosh/FLOSS studies are the most international in their coverage and cross-cultural generalization of findings.

Finally, quantitative data and analyses arising from survey research of FOSS efforts are best suited for describing frequency and distribution of univariate data, as well as correlation associations among multi-variate data that characterize FOSSD. However, these data and analyses are often comparatively weak when used to characterize the structure and performance of complex socio-technical processes whose activities, participant roles, and resources are highly situated and interdependent, yet occur in relatively low frequency and evolve over time.

Ethnographically informed methods

While survey research methods stress collection and analysis of data that is usually easy to quantify, not all phenomena operating within or around FOSS work practices, development processes, or community dynamics are readily captured or characterized in quantitative form. Thus, qualitative research methods are needed and often better suited to such discovery-oriented or participant-observer studies of FOSS development efforts [cf. Seaman 1999, Viller and Sommerville 2000]. Central to such studies are ethnographic or ethnographically informed research methods that are intended for studies where face-to-face interviews or co-located observation are central, whereas most of the action and interactions of interest in FOSSD efforts take place online across the

Internet/Web [cf. Hakken 1999, Hine 2000, Smith and Pollock 1999] in virtual organizations represented by Web sites or portals.

Qualitative ethnographic methods are better suited to the study of the structure and performance of complex work practices, community dynamics, or socio-technical development processes whose activities and participant roles are highly situated and interdependent, yet occur in relatively low frequency and evolve over time. Here there are studies by Scacchi [2002], Iannacci [2005], Elliott and Scacchi [2003, 2005, 2006], Reis and Fortes [2002], Jensen and Scacchi [2005, 2006a,b], Lanzara and Morner [2005], Longchamp [2005], Duchenaunt [2005], and Sack *et al.* [2006]. A common limitation of such studies is that they tend to focus attention to a single FOSS project setting, though this is not inherent in the method. For example, Scacchi [2002, 2004] and Jensen and Scacchi [2005] examine multiple independent FOSS project settings in order to perform comparative, cross case analyses [cf. Seaman 1999]. Similarly, these ethnographic studies tend to entail longitudinal data collection and devote particular attention to collection of FOSS development and communication artifacts, and thus employ methods for discourse and document genre analyses [cf. Kwansik and Crowston 2005, Spinuzzi 2003], as well as computational or ethnographic hypermedia analyses [Duchenaunt 2005, Jensen and Scacchi 2005, Sack *et al.* 2006, Scacchi, Jensen, *et al.* 2006]. As a result, (virtual) ethnographic studies are well-suited to small research groups who are also equipped and competent with Web-based data mining tools for searching, crawling, indexing, coding and cross-coding textual data [cf. Seaman 1999] found in FOSSD project Web sites (e.g, development artifacts or informalisms).

Mining FOSS artifact repositories and artifact analysis methods

Reflective practice, surveys, and ethnographic studies have been long employed in empirical studies of software development of all kinds. The world of FOSS does however provide a new opportunity for study that previously was unavailable or at least uncommon in the software research community. One such opportunity arises from the public accessibility of the source code and related development and communication artifacts associated with FOSS project Web sites or FOSS community repositories or portals like SourceForge.org and others [cf. Harrison 2001].

The accessibility of the source code and artifacts means that they can be directly subjected to various kinds of automated or semi-automated processing techniques, including text data mining, crawling and indexing, statistical analyses, and machine learning. These processing techniques give rise to not only new ways and means for analyzing large textual FOSS data sets, but also to investigate research questions or problems that heretofore could not be addressed with the established research methods for studying software development. For example, there are now studies of FOSS source code that reveal patterns of the growth and evolution of different FOSS systems over time, [Capiluppi *et al.* 2003, Schach, *et al.* 2002, Paulson, *et al.* 2004, Smith, *et al.* 2004].

Common among the findings in these studies is growing evidence for sustained exponential growth rates for large, highly successful FOSS systems [cf. Scacchi 2006a], though the majority of FOSS projects fail to grow at all [cf. Madey *et al.* 2002, 2005]. Such findings stand in contrast to the established wisdom from long-standing studies of software evolution in the world of traditional (closed-source) software, where inverse-square growth rates are more common observed [cf. Lehman 1980, 2002].

Other studies of FOSS repositories have focused attention to (textual) artifacts associated with different FOSS projects. For example, in a widely cited study of the development of the Apache Web server and Mozilla Web browser, Mockus, Fielding and Herbsleb [2002] reported that they were able to investigate, extract, and quantify modification requests captured in change logs and bug reporting repositories associated with each of these two projects. They analyzed and compared their findings on bug frequency and severity over time identified in modification requests for the browser and server, with those found in commercial (proprietary) telecommunications systems software. Subsequently, they found these FOSS projects produce comparable or higher quality software, but without the software project management regimen used in industry.

Elsewhere, Madey, *et al.* [2002, 2005] and Lopez-Fernandez, *et al.* [2004, 2006] employ data mining techniques to extract and analyzing social network relationships between developers who communicate with each other in the course of modifying or updating FOSS project source code in stored in common transactional repositories like CVS [Fogel 1999]. Figure 3 from Madey and colleagues displays how a small number of FOSS developer can establish social network links through computer-mediated messaging that connect developers spanning multiple FOSS projects together. This helps create critical mass [Marwell and Oliver 1993] that sustains their collective FOSS development efforts. However, if the linchpin developers were missing, then the multi-project cluster may dissociate or fail to link up, resulting in an insufficient collective social mass needed to go critical and enable network externalities like exponential growth of community source code. Crowston and Howison [2006] similarly demonstrate how FOSS development teams often self-organize into a team hierarchy, where a small number of core developers

serve as the critical center of gravity for a larger community of contributors and end-users.

Last, Ripoche and Gasser [2003] demonstrate how automated mining of textual and transaction data entered into a FOSS bug tracking system (e.g., Bugzilla) can be used to extract and generate a model of the bug management process, and how it serves to help maintain and evolve the design of a FOSS system like the Mozilla Web browser.

Overall, FOSS source code and artifact repositories offer a vast array of textual and transactional data that is just beginning to be explored. For example, FOSS project meta-data is now being collected with new Web sites emerging (e.g., FLOSSmole [Howison *et al.* 2006] at ossmole.sourceforge.net; also see www.ohloh.net) that organize and provide access these data. This contributes to an open, shared research infrastructure for studying FOSS socio-technical characteristics, structures, and dynamics across potentially a very large sample of FOSS projects that can be analyzed quantitatively and textually. Further, as these studies employ automated tools for data collection, coding, and analysis, then these methods for mining FOSS repositories become increasingly accessible to small research groups or individual FOSS scholars. However, data in FOSS repositories like change logs [Chen *et al.* 2004] or modification requests associated with source code updates entered into CVS repositories [German 2006] require careful review, cleaning, and normalization (e.g., dealing with missing or overloaded data records). Thus, mining FOSS repositories does require care and attention to both the data and their analysis, since (a) data quality problems abound which require explicit attention (especially in publication), (b) researchers may not have first-hand experience in using these

repositories as FOSSD project participants, and (c) these repositories were not conceived or intended to be used for collecting data on FOSSD practices or processes.

Methods for mining FOSS repositories also offer the potential for either/both in-depth (e.g., project specific) and in-breadth (scalable to large samples of projects) empirical studies of FOSS development efforts. Thus, expect to see analysis of FOSS project source or artifacts increasingly dominating large-scale quantitative studies of software development of any kind, by research groups that include experts and emerging scholars (e.g., graduate or post-doctoral students) who are motivated to develop and apply new textual data or Web mining tools/techniques to established FOSSD repositories of various kinds supporting different kinds of development activities or communities.

Multi-modal modeling and analysis of FOSS socio-technical interaction networks

One other research method being used to study FOSS projects that is starting to gain some traction involves use of hybrid schemes involving multiple research methods. Two such efforts are those of Duchenaunt, Sack and colleagues [Duchenaunt 2005, Sack *et al.* 2006], and Scacchi and associates [Jensen and Scacchi 2005, Scacchi, Jensen, *et al.* 2006]. Both of these respective efforts focus on collection of ethnographic data of socio-technical interaction networks or processes [cf. Duchenaunt 2005, Scacchi 2005] they discover in the FOSS projects identified in their studies, using virtual ethnographic techniques and computational data mining, modeling, and visualization tools. In a sense, these multi-modal research methods seek to pull together the robust qualitative field study methods used in ethnographic studies together with techniques employing automated or

semi-automated data mining and validation tools in ways that can be put into action by a small research group. However, these multi-modal methods have not yet been applied to large samples of FOSS projects, and thus it is unclear whether such methods can scale up to such challenge, or whether some other mix of research methods will be needed.

Discussion

One of the defining characteristics of data about the FOSSD projects is that in general it is publicly available on a global basis [Harrison 2001, Scacchi 2006a]. Data about FOSSD products, artifacts, and other resources is kept in repositories associated with a project's Web site. This may include the site's content management system, computer mediated communication systems (email, persistent chat facilities, and discussion forums), software versioning or configuration management systems, and networked file systems. FOSSD process data is generally either extractable or derivable from data/content in these artifact repositories. First-person data may also be available to those who participate in a project, even if just to remotely observe (“lurk”) or to electronically interview other participants about development activities, tools being used, the status of certain artifacts, and the like. The availability of such data perhaps suggest the a growing share of empirical software engineering research will be performed in the domain of FOSSD projects, rather than using traditional sources of data from in-house or proprietary software development projects. These traditional non-FOSS projects will continue to have constraints on access and publication. FOSSD process data collection from publicly accessible artifact repositories may also be found to be more cost-effective compared to studies of traditional closed-source, proprietary, and in-house software development repositories [cf. Cook 1998].

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 observe, 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 real-

world 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. FOSSD 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.
- Capiluppi, 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.

Crowston, K. and Howison, J., (2006). Hierarchy and centralization in free and open source software team communications, *Knowledge Technology & Policy*, 18(4), Winter, 65-85.

Crowston, K., Howison, J., and Annabi, H., (2006). Information systems success in free and open source software development: theory and measures, *Software Process—Improvement and Practice*, 11(2), 123-148.

Crowston, K., and Scozzi, B., (2002). Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings--Software*, 149(1), 3-17.

Curtis, B., Krasner, H., and Iscoe, N., (1988). A Field Study of the Software Design Process for Large Systems, *Communications ACM*, 31(11), 1268-1287.

Danziger, J., (1979). The Skill Bureaucracy and Intraorganizational Control: The Case of the Data-Processing Unit, *Sociology of Work and Occupations*, 21(3), 206-218.

De Souza, C. R. B., Froehlich, J., and Dourish, P. (2005) Seeking the Source: Software Source Code as a Social and Technical Artifact. *Proc. ACM Intern. Conf. Supporting Group Work (GROUP 2005)*, 197-206, Sanibel Island, Florida,

DiBona, C., Cooper, D., and Stone, M., (2005). *Open Sources 2.0*, O'Reilly Media, Sebastopol, CA.

DiBona, C., Ockman, and Stone, M., (1999). *Open Sources: Voices from the Open Source Revolution*, O'Reilly Media, Sebastopol, CA.

Ducheneaut, N. (2005). Socialization in an Open Source software community: A socio-technical analysis. *Computer Supported Cooperative Work*, 14(4), 323-368.

Ebenau, R.G. and Strauss, S.H. (1994). *Software Inspection Process*. McGraw-Hill, New York.

Elliott, M.S., (2006). Examining The Success of Computerization Movements in the Ubiquitous Computing Era: Free and Open Source Software Movements, in K.L. Kraemer and M. Elliott (eds.), *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, Information Today, Inc., to appear.

Elliott, M. and Scacchi, W., (2003). Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration, *Proc. ACM Intern. Conf. Supporting Group Work*, 21-30, Sanibel Island, FL, November.

Elliott, M. and Scacchi, W., (2005). Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, 152-172, Idea Group Publishing, Hershey, PA.

- Elliott, M. and Scacchi, W., (2006), Mobilization of Software Developers: The Free Software Movement, (submitted for publication).
- Erenkrantz, J., (2003). Release Management within Open Source Projects, *Proc. 3rd. Workshop on Open Source Software Engineering*, 25th. Intern. Conf. Software Engineering, Portland, OR, May.
- Erickson, T., (2000). Making Sense of Computer-Mediated Communication (CMC): CMC Systems as Genre Ecologies, *Proc. 33rd Hawaii Intern. Conf. Systems Sciences*, IEEE Press, 1-10, January.
- Espinosa, J. A., Kraut, R.E., Slaughter, S. A., Lerch, J. F., Herbsleb, J. D., Mockus, A. (2002). Shared Mental Models, Familiarity, and Coordination: A Multi-method Study of Distributed Software Teams. *Intern. Conf. Information Systems*, 425-433, Barcelona, Spain, December.
- Feller, J., and Fitzgerald, B., (2002). *Understanding Open Source Software Development*, Addison-Wesley, NY.
- Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K. (eds.), (2005). *Perspectives on Free and Open Source Software*, MIT Press, Cambridge, MA.
- Fielding, R.T., (1999). Shared Leadership in the Apache Project. *Communications ACM*, 42(4), 42-43.
- Fink, M., (2003). *The Business and Economics of Linux and Open Source*, Prentice Hall PTR, Upper Saddle, NJ.
- Fischer, G., (2001). External and shareable artifacts as opportunities for social creativity in communities of interest, in J. S. Gero and M. L. Maher (eds), *Proc. Computational and Cognitive Models of Creative Design*, 67-89, Heron Island, Australia, December.
- FLOSS (2002). *Free/Libre and Open Source Software: Survey and Study*, FLOSS Final Report, <http://www.flossproject.org/report/> (accessed July 2006).
- Fogel, K., (1999). *Open Source Development with CVS*, Coriolis Press, Scottsdale, AZ.
- Fogel, K. (2005). *Producing Open Source Software: How to Run a Successful Free Software Project*, O'Reilly Press, Sebastopol, CA.
- Gay, J. (ed.), (2002). *Free Software Free Society: Selected Essays of Richard M. Stallman*, GNU Press, Free Software Foundation, Boston, MA.
- Gacek, C. and Arief, B., (2004). The Many Meanings of Open Source, *IEEE Software*, 21(1), 34-40, January/February.

- Gallivan, M., (2001). Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies, *Information Systems J.*, 11(4), 277-304.
- German, D., (2003). The GNOME Project: A case study of open source, global software development, *Software Process—Improvement and Practice*, 8(4), 201-215.
- German, D. (2006). An Empirical Study of Fine-Grained Software Modifications, *Empirical Software Engineering*, 11(3), 369-393, 2006.
- Ghosh, R., (ed.), (2005). *CODE: Collaborative Ownership and the Digital Economy*, MIT Press, Cambridge, MA.
- Ghosh, R. and Prakash, V.V., (2000). The Orbiten Free Software Survey, *First Monday*, 5(7), July, http://www.firstmonday.org/issues/issue5_7/ghosh/index.html accessed 1 June 2006.
- Goldman, R. and Gabriel, R.P., (2005). *Innovation Happens Elsewhere: Open Source as Business Strategy*, Morgan Kaufmann Publishers, San Francisco, CA.
- Greene, T.C., (2001). Ballmer: “Linux is a Cancer”, *The Register*, http://www.theregister.co.uk/2001/06/02/ballmer_linux_is_a_cancer/, 2 June 2001.
- Grinter, R. E. (1996). Supporting Articulation Work Using Configuration Management Systems, *Computer Supported Cooperative Work: The Journal of Collaborative Computing*. 5(4): 447-465.
- Hann, I-H., Roberts, J., Slaughter, S., and Fielding, R., (2002). Economic Incentives for Participating in Open Source Software Projects, in *Proc. Twenty-Third Intern. Conf. Information Systems*, 365-372, December.
- Hars, A. and Ou, S., (2002). Working for Free? Motivations for participating in open source projects, *Intern. J. Electronic Commerce*, 6(3).
- Hakken, D. (1999). *Cyborgs@Cyberspace? An Ethnographer Looks at the Future*, Routledge, London.
- Harrison, W., (2001). Editorial: Open Source and Empirical Software Engineering, *Empirical Software Engineering*, 6(2), 193-194.
- Hertel, G., Neidner, S., and Hermann, S., (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel, *Research Policy*, 32(7), 1159-1177, July.
- Hertzum, M. (2002). The importance of trust in software engineers' assessment and choice of information sources, *Information and Organization*, 12(1), 1-18.

- Hine, C.M. (2000). *Virtual Ethnography*, Sage Publications, Newbury Park, CA.
- Howison, J., Conklin, M., and Crowston, K. (2006). FLOSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. *Intern. J. Info. Tech. And Web Engineering*, 1(3), 17-26.
- Huntley, C.L., (2003). Organizational Learning in Open-Source Software Projects: An Analysis of Debugging Data, *IEEE Trans. Engineering Management*, 50(4), 485-493.
- Iacono, C.S. and Kling, R., (2001). Computerization Movements: The Rise of the Internet and Distant Forms of Work, in Yates, J.A., and Van Maanen, J. (Eds.), *Information Technology and Organizational Transformation: History, Rhetoric, and Practice*, Sage Publications, Newbury Park, CA.
- Iannacci, F. (2005a). Coordination Processes in Open Source Software Development: The Linux Case Study, *Emergence: Complexity & Organization (E:CO)* , 7(2), 21-31.
- Iannacci, F. (2005b). Beyond Markets and Firms: The Emergence of Open Source Networks, *First Monday*, 10(5).
- Jensen, C. and Scacchi, W., (2004). Collaboration, Leadership, and Conflict Negotiation in the NetBeans.org Community, *Proc. 4th Workshop on Open Source Software Engineering*, Edinburgh, UK, May.
- Jensen, C. and Scacchi, W., (2005). Process Modeling Across the Web Information Infrastructure, *Software Process—Improvement and Practice*, 10(3), 255-272, July-September.
- Jensen, C. and Scacchi, W. (2006a). Discovering, Modeling, and Reenacting Open Source Software Development Processes, in S.T. Acuna and M.I. Sanchez-Segura (eds.), *New Trends in Software Process Modeling*, Series in Software Engineering and Knowledge Engineering, Vol. 18, 1-20, World Scientific Publishing, Singapore.
- Jensen, C. and Scacchi, W., (2006b). Modeling Recruitment and Role Migration Processes in Open Source Software Development Projects, submitted for publication, April.
- Kim, A.J., (2000). *Community-Building on the Web: Secret Strategies for Successful Online Communities*, Peachpit Press.
- Kling, R., and Scacchi, W. (1982). The Web of Computing: Computer Technology as Social Organization, in M.C. Yovits (ed.), *Advances in Computers*, 21, 1-90.
- Koch, S. (Ed.), (2005). *Free/Open Source Software Development*, Idea Group Publishing, Hershey, PA.

Kwansik, B. and Crowston, K., (2005). Introduction to the special issue: Genres of digital documents, *Information, Technology and People*, 18(2).

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

Lanzara, G.F. and Morner, M., (2005). Artifacts rule! How organizing happens in open source software projects, in B. Czarniawska and T. Hernes (eds.), *Actor-Network Theory and Organizing*, pp. 67-90, Liber & Copenhagen Business School Press, Malmo, Sweden.

Lave, J. and Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*, Cambridge University Press, Cambridge, UK.

Lehman, M.M., (1980). Programs, Life Cycles, and Laws of Software Evolution, *Proc. IEEE*, 68, 1060-1078.

Lehman, M.M., (2002). Software Evolution, in J. Marciniak (ed.), *Encyclopedia of Software Engineering*, 2nd Edition, John Wiley and Sons, Inc., New York, 1507-1513, 2002. Also see "Software Evolution and Software Evolution Processes," *Annals of Software Engineering*, 12, 275-309, 2002.

Lerner, J. and Tirole, J., (2002). Some Simple Economics of Open Source, *J. Industrial Economics*, 50(2), 197-234.

Lessig, L. (2000). *Code and other Laws of Cyberspace*, Basic Books, New York.

Lessig, L. (2005). *Free Culture: The Nature and Future of Creativity*, Penguin, New York.

Longman, J. (2005). Open Source Software Development Process Modeling, in S.T. Acuña and N. Juristo (eds.), *Software Process Modeling*, 29-64, Springer Science+Business Media Inc., New York.

Ljungberg, J., (2000). Open Source Movements as a Model for Organizing, *European J. Info. Sys.*, 9(4), 208-216.

Lopez-Fernandez, L., Robles, G., and Gonzalez-Barahona, J.M., (2004). Applying Social Network Analysis to the Information in CVS Repositories, *Proc. First Intern. Workshop on Mining Software Repositories*, 101-105, Edinburgh, UK, May.

Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J.M., and Herraiz, I. (2006). Applying Social Network Analysis to Community-Drive Libre Software Projects, *Intern. J. Info. Tech. and Web Engineering*, 1(3), 27-28.

Madey, G., Freeh, V., and Tynan, R., (2002). The Open Source Development Phenomenon: An Analysis Based on Social Network Theory, *Proc. Americas Conf. Info. Systems (AMCIS2002)*, 1806-1813, Dallas, TX.

Madey, G., Freeh, V., and Tynan, R., (2005). Modeling the F/OSS Community: A Quantitative Investigation, in S. Koch (ed.), *Free/Open Source Software Development*, 203-221, Idea Group Publishing, Hershey, PA.

Marwell, G. and Oliver, P., (1993). *The Critical Mass in Collective Action: A Micro-Social Theory*. Cambridge University Press, Cambridge, England.

MITRE Corporation, (2003). *Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense*, January, <http://www.egovos.org/pdf/dodfoss.pdf>.

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

Monge, P.R., Fulk, J., Kalman, M.E., Flanagan, A.J., Parnassa, C., and Rumsey, S., (1998). Production of Collective Action in Alliance-Based Interorganizational Communication and Information Systems, *Organization Science*, 9(3), 411-433.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y., (2002). Evolution Patterns of Open-Source Software Systems and Communities, *Proc. 2002 Intern. Workshop Principles of Software Evolution*, 76-85.

Noll, J. and Scacchi, W., (1999). Supporting Software Development in Virtual Enterprises, *J. Digital Information*, 1(4), February, <http://jodi.tamu.edu/Articles/v01/i04/Noll/>.

O'Dowd, D. (2004). No Defense for Linux: Inadequate Security Poses National Security Threat, *Design News*, 19 July 2004. <http://www.designnews.com/article/CA435615.html>

Olson, M., (1971). *The Logic of Collective Action*, Harvard University Press, Cambridge, MA.

O'Mahony, S. (2003). Guarding the Commons: How Community Managed Software Projects Protect their Work, *Research Policy* 32(7), July, 1179-1198.

OSBC, (2006). Open Source Business Conference, <http://www.osbc.com>. (accessed 15 July 2006).

Ostrom, E., Calvert, R., and T. Eggertsson (eds.), (1990). *Governing the Commons: The Evolution of Institutions for Collective Action*, Cambridge University Press, Cambridge, England.

- Ovaska, P., Rossi, M. and Marttiin, P. (2003). Architecture as a Coordination Tool in Multi-Site Software Development, *Software Process—Improvement and Practice*, 8(3), 233-247.
- Paulson, J.W., Succi, G., and Eberlein, A., (2004). An Empirical Study of Open-Source and Closed-Source Software Products, *IEEE Trans. Software Engineering*, 30(4), 246-256, April.
- Pavelicek, R., (2000). *Embracing Insanity: Open Source Software Development*, SAMS Publishing, Indianapolis, IN.
- Payton, S., Herz, J.C., Lucas, M. and Scott, J. (2006). *Open Technology Development: Roadmap Plan*, Final Report, Advanced Systems & Concepts, Deputy Undersecretary of Defense, <http://www.acq.osd.mil/asc>, April 2006. (accessed 15 August 2006).
- Porter, A.A., Siy, H.P., Toman, C.A. & Votta, L.G., (1997). An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development. *IEEE Trans. on Software Engineering*, 23, 329-346.
- Porter, A.A., Yilmaz, C., Memon, A.M., Krishna, A.S., Schmidt, D.C., and Gokhale, A., (2006). Techniques and Processes for Improving the Quality and Performance of Open-Source Software, *Software Process—Improvement and Practice*, 11(2), 163-176.
- Preece, J., (2000). *Online Communities: Designing Usability, Supporting Sociability*. Chichester, UK: John Wiley & Sons, New York.
- Reis, C.R. & Fortes, R.P.M., (2002). An Overview of the Software Engineering Process and Tools in the Mozilla Project, *Proc. Workshop on Open Source Software Development*, Newcastle, UK, February.
- Ripoche, G. and Gasser, L., (2003). Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair, *Proc. 16th Intern. Conf. Software Engineering & its Applications* (ICSSEA-03), Paris, France, December, 2003.
- Sack, W., Detienne, F., Ducheneaut, Burkhardt, Mahendran, D., and Barcellini, F., (2006). A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software, *Computer Supported Cooperative Work*, (to appear).
- Sawyer, S., (2001). Effects of intra-group conflict on packaged software development team performance, *Information Systems J.*, 11, 155-178, 2001.
- Scacchi, W., (2002). Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39, February.
- Scacchi, W., (2004). Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February.

- Scacchi, W., (2005). Socio-Technical Interaction Networks in Free/Open Source Software Development Processes, in S.T. Acuña and N. Juristo (eds.), *Software Process Modeling*, 1-27, Springer Science+Business Media Inc., New York.
- Scacchi, W., (2006a). Understanding Free/Open Source Software Evolution, in N.H. Madhavji, M.M. Lehman, J.F. Ramil and D. Perry (eds.), *Software Evolution*, John Wiley and Sons Inc, New York, to appear.
- Scacchi, W., (2006b). Emerging Patterns of Intersection and Segmentation when Computerization Movements Interact, in K.L. Kraemer and M. Elliott (eds.), *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, Information Today, Inc., to appear.
- Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K. (2006). Understanding Free/Open Source Software Development Processes, *Software Process--Improvement and Practice*, 11(2), 95-105, March/April.
- Scacchi, W., Jensen, C., Noll, J., and Elliott, M., (2006), Multimodal Modeling, Analysis, and Validation of Open Source Software Development Processes, *Intern. J. Information Technology and Web Engineering*, 1(3), 49-63.
- Schach, S.R., Jin, B., Wright, D.R., Heller, G.Z., and Offutt, A.J., (2002). Maintainability of the Linux Kernel, *IEE Proceedings – Software*, 149(1), 18-23, February.
- Seaman, C.B., (1999). Qualitative Methods in Empirical Studies of Software Engineering, *IEEE Trans. Software Engineering*, 25(4), 557-572, July/August.
- Seaman, C.B. and Basili, V. (1998). Communication and Organization: An Empirical Study of Discussion in Inspection Meetings, *IEEE Trans. Software Engineering*, 24(6), 559-572, July.
- Sharma, S., Sugumaran, and Rajagopalan, B., (2002). A Framework for Creating Hybrid Open-Source Software Communities, *Information Systems J.*, 12(1), 7-25.
- Sim, S.E. and Holt, R.C., (1998). The Ramp-Up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize, *Proc. 20th Intern. Conf. Software Engineering*, Kyoto, Japan, 361-370, 19-25 April.
- Smith, M. and Kollock, P. (eds.), (1999). *Communities in Cyberspace*, Routledge, London.
- Smith, N., Capiluppi, A. and Ramil, J.F., (2004). Qualitative Analysis and Simulation of Open Source Software Evolution, *Proc. 5th Software Process Simulation and Modeling Workshop (ProSim '04)*, Edinburgh, Scotland, UK, May.

- Snow, D.A., Soule, S.A., and Kriesi, H., (2004), *The Blackwell Companion to Social Movements*, Blackwell Publishers Ltd., Victoria, Australia.
- Spinuzzi, C., (2003). *Tracing Genres through Organizations: A Sociocultural Approach to Information Design*, MIT Press, Cambridge, MA.
- Sommerville, I., (2004). *Software Engineering, 7th Edition*, Addison-Wesley, New York.
- Stewart, K.J. and Gosain, S., (2001). An Exploratory Study of Ideology and Trust in Open Source Development Groups, *Proc. 22nd Intern. Conf. Information Systems (ICIS-2001)*, in New Orleans, LA.
- Truex, D., Baskerville, R., and Klein, H., (1999). Growing Systems in an Emergent Organization, *Communications ACM*, 42(8), 117-123.
- Viller, S. and Sommerville, I., (2000). Ethnographically informed analysis for software engineers, *Intern. J. Human-Computer Studies*, 53, 169-196.
- von Hippel, E., and von Krogh, G., (2003). Open Source Software and the “Private-Collective” Innovation Model: Issues for Organization Science, *Organization Science*, 14(2), 209-223.
- von Krogh, G., Spaeth, S., and Lakhani, K., (2003). Community, Joining, and Specialization in Open Source Software Innovation: A Case Study, *Research Policy*, 32(7), 1217-1241, July.
- Weber, S., (2004), *The Success of Open Source*, Harvard University Press, Cambridge, MA.
- West, J. and O’Mahony, S., (2005). Contrasting Community Building in Sponsored and Community Founded Open Source Projects, *Proc. 38th. Hawaii Intern. Conf. Systems Sciences*, Waikola Village, HI.
- West, J. and Dedrick, J., (2006). The Effect of Computerization Movements Upon Organizational Adoption of Open Source, tin K.L. Kraemer and M. Elliott (eds.), *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, Information Today, Inc., to appear.
- Wheeler, D.A., (2005). Why Open Source Software / Free Software (OSS/FS, FLOSS or FOSS)? Look at the Numbers, http://www.dwheeler.com/oss_fs_why.html , Accessed 15 November 2005.
- Williams, S., (2002). *Free as in Freedom: Richard Stallman's Crusade for Free Software*, O'Reilly Books, Sebastopol, CA.

Yamauchi, Y., Yokozawa, M., Shinohara, T., and Ishida, T., (2000). Collaboration with Lean Media: How Open-Source Software Succeeds, *Proc. Computer Supported Cooperative Work Conf. (CSCW'00)*, 329-338, Philadelphia, PA, ACM Press, December.

Ye, Y., Nakajoki, K., Yamamoto, Y., and Kishida, K., (2005). The Co-Evolution of Systems and Communities in Free and Open Source Software Development, in S. Koch (ed.), *Free/Open Source Software Development*, 59-82, Idea Group Publishing, Hershey, PA.

Ye, Y. & Kishida, K., (2003). Towards an understanding of the motivation of open source software developers, *Proc. 25th Intern. Conf. Software Engineering*, Portland, OR, 419-429, IEEE Computer Society, May.