ITR: Understanding Open Software Communities, Processes and Practices: A Socio-Technical Approach

Final Report

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
wscacchi@uci.edu, jnoll@scu.edu
June 2005

The research results documented in this final report benefited from a grant #0083075 from the National Science Foundation. No endorsement implied.

Summary

This report documents the results, findings, and research methods employed in examining issues in the management of knowledge-intensive distributed systems in a research project funded by through the National Science Foundation during the period of September 2000- June 2005. The project was based at the University of California, Irvine Institute of Software Research (www.isr.uci.edu), though the research effort was performed through collaborative studies conducted at ISR and at the Santa Clara University. The project was directed by Dr. Walt Scacchi at ISR in collaboration with Dr. Mark Ackerman, University of Michigan, Ann Arbor and ISR.

This project focused on developing empirically grounded models and theories of the social processes, technical system configurations, organizational contexts, and interrelationships that give rise to open source software. OSS represented a comparatively new approach for communities of like-minded participants to develop software systems that are intended to be shared freely, rather than offered as commercial products. While there is a growing popular literature attesting to open software, there is little in the way of careful systematic empirical study that informs how such communities produce software; how they coordinate software development across different settings; and what social processes, work practices, and organizational contexts are necessary to their success. Thus, to the extent that science research communities and commercial enterprises seek the supposed efficacy of open software, they will need better grounded theories of use to allow effective investment of their resources. This study investigated four communities engaged in developing OSS. Ethnographic field study methods were employed to examine each community from both a technical and social viewpoint. Studying both social and technical arrangements allowed the examination of the continual emergence of both within a joint social-technical ecology. Case study methods were used to compare emerging findings across communities.

The report is organized into four major sections, following a brief introduction. Each section contains four chapters that represent and document our approach and the results we produced. Each chapter is each of the four sections is available for download and review on the project's Web site, http://www.isr.uci.edu/research-open-source.html. The first section provides an overview that starts from our proposed effort, followed by an example study that utilizes the various concepts, techniques, and tools that we investigated throughout the project.

The second section examines our approach to discovering processes from examination of open source software development (OSSD) projects within or across different project communities.

The third section examines our approach to developing tools and techniques for computationally modeling the processes that have been discovered.

The fourth and final section examines studies of how "invisible" OSSD processes can breakdown or fail due to unmanaged conflicts, lack of appropriate leadership and coordination mechanisms that rely on internal or external communication systems,

discourse artifacts, and other objects that span the boundaries that separate the distributed and hidden workflows of people doing knowledge-intensive software development tasks. It also presents results from two recent studies that examine how OSSD processes can coalesce into larger social movements or "computerization movements."

Introduction

OSSD projects are knowledge-intensive efforts that are dispersed across participants in multiple locations working at different times in a loosely-coupled or mostly autonomous manner. OSSD projects are typically self-organizing and are at best weakly coordinated by a corporate sponsor, when such a sponsor exists. But in projects like NetBeans.org, Apache.org, Eclipse.org, and Mozilla.org, they can involve the active contribution of hundreds or thousands of knowledge workers who collectively act to develop complex software systems and related development documents or artifacts. Global OSSD projects are thus quintessential endeavors that are poorly understood, yet produce complex knowledge-intensive products (OSSD systems and development artifacts) through unseen or hidden workflows that are potentially globally dispersed.

The materials in this report start from what we initially proposed as our approach for how to conduct this study. This is described in a chapter titled, Software Development Practices in Open Software Development Communities. The vision outlined proscribes an ambitious multi-year study, though the actual effort was limited to a four year effort. This chapter is complemented by the second examining how to Understand the Social, Technological, and Policy Implications of OSSD. This chapter investigates how the results from the overall research study can be framed to address issues in public policy, particularly in science and research policy, such as might be of interest to national research funding agencies or foundations like the NSF. Thus a foundation for further study in this area has been established. The third chapter develops a systematic, empirically grounded approach for Understanding the Requirements for Developing Open Source Software Systems. This chapter represents one of the first, if not the first, empirical studies of OSSD practices across different OSSD project communities in terms of a widespread but often poorly understood process for developing software system requirements. This chapter thus compares and contrasts how OSSD processes are different from those traditionally attributed to the development of commercial or closed-source software systems. The last chapter provides the results of an extensive survey of published studies of OSSD projects that focuses analytical attention to characterizing the forms and diversity of socio-technical processes that now characterize how OSSD projects build software.

The second section examines and compares the practices and processes of OSSD found within or across different OSSD project communities. The first chapter examines When is Free/Open Source Software Development *Faster*, *Better*, and *Cheaper* than Software Engineering? This chapter further compares and contrasts the practices that characterize OSSD projects with those associated with software engineering. The second chapter examines Free/Open Source Software Development Practices in the Computer Game

Community. This chapter identifies how OSSD practices and processes have infiltrated the world of networked computer game development, and for what appears to be the betterment of the computer game industry. The third chapter provides a similar kind of study but within the world of OSSD and E-Commerce, in the study of Open EC/B: Electronic Commerce and Free/Open Source Software Development. Here, we examine a project community where one might be surprised to find the advocates of "free and open" software development working on applications that perhaps are traditionally viewed as being primarily conceived as "costly but valuable, and closed." The last chapter in this section examines a new approach to Understanding Free/Open Source Software Evolution. This is based on a survey of empirical studies of OSSD patterns across multiple releases over time. It reports and examines the exponential or super-linear growth of OSS systems that are in widespread and popular use. The chapter offers some initial interpretations as to why such exponential growth might be possible for OSS systems, but unlikely for conventional closed source software, or for development projects that follow traditional software engineering principles and practices.

The third section examines issues arising from the modeling of processes, whether these processes are independently constructed, or else derived via process discovery. The seventh chapter then employs these modeling languages and techniques to examine the application of an experimental approach to Modeling Recruitment and Role Migration Processes in OSSD Projects. These languages and techniques are similarly applied in another related effort that focuses on Process Modeling Across the Web Information Infrastructure in the second chapter in this section. This paper represents a significant research result in being the first such model that accounts for the ongoing development and evolution of a complex knowledge-intensive virtual enterprise of autonomous projects that collectively are responsible for the much of the core information infrastructure of the World Wide Web. The third chapter describes and summarizes our collective Experiences in Discovering, Modeling, and Re-enacting OSSD Processes. The fourth and last chapter in this section provides an overall summary of our research methods and results for how to discover, model, analyze, and enact (or re-enact) distributed processes (derived from implicit or invisible workflows) that span a globally dispersed, large-scale, corporate-sponsored open source software development project like NetBeans.org. Emphasis here is directed at the example case of that proposes a multi-modal approach to modeling the previously hidden process that determines how the NetBeans.org enterprise conducts its "requirements and release" activities associated with the periodic release of a new version of the NetBeans software system. These results are found in a chapter titled, Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes. This chapter and the one that precedes it thus provide an overview of where the project started and the kind of empirical results that we could document and demonstrate by the end of the project period.

The last section of this report begins to explore how knowledge-intensive processes and sometimes hidden OSSD workflows can breakdown, fail, or other disarticulate in the course of their enactment. One reason processes can fail is when the details and understanding of the roles people play in their enactment, the tools and resources they employ in performing the enactment, are hidden or unclear to others either upstream or

downstream of the process workflow. The first chapter examines these issues using a case study of Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture. The ethnographic results from this empirical study help lay the foundation for recognizing how beliefs, values, and norms that people do (or do not) share in the course of their knowledge work shapes the actions they take in accomplishing their work and workflow, in ways that subsequently may be visible or invisible (i.e., hidden) from the perspective of others. This insight in turn was then examined in a follow-up case study of a large-scale multi-site OSSD project, in the second chapter of this section in a paper titled, Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Software Development Community. The third and final chapters take a broader perspective on OSSD processes articulate, intersect, combine, and segment/breakdown across the larger socio-technical worlds in which they are found. The third chapter examines the Mobilization of Software Developers in The Free Software Movement. The fourth and final chapter provides an expansive study of OSSD as both a social movement and a computerization movement, within and across the worlds of computer game and scientific grid computing, in the study of Emerging Patterns of Intersection and Segmentation when Computerization Movements Interact.

Overall, these sixteen chapters provide a thorough documentation of the research project we engaged from Fall 2000 through Spring 2005. Our assessment is that the research effort met and exceeded what was originally proposed, given the resources provided to conduct the proposed effort. However, it is also our view that much work remains to be performed and completed before we would consider this line of research and development of an approach to managing knowledge-intensive distributed systems of work that span multiple organizations or project sites finished. Thus, this report serves to document the overall state of a significant work in progress, with further research and support needed to achieve its full potential.

In closing, the interested reader is encouraged to access, download, study, share, and build on the results of the studies included in this report from the Web site for this research project at http://www.isr.uci.edu/research-open-source.html. That site also contains many other research papers and related source of information for those who seek to engage in systematic study of the world of free/open source software development practices, processes, and community projects.

Overview

This section contains the following four chapters. The first is a simplified version of the original proposal that gave rise to this research effort.

The second chapter provides a framing of the public policy issues arising from the study of open source software projects examined in this research effort.

The third chapter represents the first major finding arising from the research effort, which reports on the discovery and practice of "software informalisms" as previously unrecognized modes of communication and interaction that supports the practice of open source software development, within and across different open source software projects. This paper has become the most widely cited research finding that has resulted from this research effort.

The fourth and last chapter in this section provides an overall survey and summary of major findings arising from systematic empirical study of free/open source software development projects during 2000-2005, from studies conducted in various international research centers. This chapter thus represents a major summary of what scientific knowledge has been developed through five years of intense, international study of the practices, processes, and communities associated with different F/OSSD projects.

Walt Scacchi, Software Development Practices in Open Software Development Communities, presented at the *1st. Workshop on Open Source Software Engineering*, Toronto, Ontario, May 2001.

Walt Scacchi, Understanding the Social, Technological, and Policy Implications of Open Source Software Development, *NSF Workshop on Open Source Software*, Arlington, VA, January 2002 (revised August 2002).

Walt Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39, February 2002.

Walt Scacchi, Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim Lakhani, Understanding Free/Open Source Software Development Processes, *Software Process--Improvement and Practice*, to appear, 2006.

Software Development Practices in Open Software Development Communities: A Comparative Case Study

(Position Paper)

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425 USA
http://www.ics.uci.edu/~wscacchi
wscacchi@ics.uci.edu

Overview

This study presents an initial set of findings from an empirical study of social processes, technical system configurations, organizational contexts, and interrelationships that give rise to open software. "Open software", or more narrowly, open source software, represents an approach for communities of like-minded participants to develop software system representations that are intended to be shared freely, rather than offered as closed commercial products. While there is a growing popular literature attesting to open software [DiBona, Ockman, Stone 1999, Fogel 1999], there are very few systematic studies [e.g., Feller and Fitzgerald 2000, Mockus, Fielding, Herbsleb 2000] that informs how these communities produce software. Similarly, little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success. To the extent that academic research communities and commercial enterprises seek the supposed efficacy of open software [Smarr and Graham 2000], they will need grounded models of the processes and practices of open software development to allow effective investment of their resources. This study investigates four communities engaged in open software development. Case study methods are used to compare practices across communities.

Understanding open software development practices

Our interest is in understanding the practices and processes of open software development in different communities. We assume there is no prior model or globally accepted framework that defines how open software is or should be developed. So our starting point is to investigate open software practices in different communities to be able to identify what communities members believe are their best practices.

We have chosen four different communities to study. These are those centered about the development of software for:

- *Networked computer game worlds*--first person shooters (e.g., Quake Arena, Unreal Tournament), massively multiplayer online role-playing games (MMORPG, e.g., Everquest, Ultima Online), and others (e.g., The Sims (maxis.com), Neverwinter Nights (bioware.com))

- *Internet infrastructure*--e.g., Apache web server (www.apache.org), InterNet News, Mozilla Web browser, etc.
- *X-ray astronomy and deep space imaging*--e.g., Chandra X-Ray Observatory (http://asc.harvard.edu/swapmeet_top.html) and the European Space Agency's XMM-Newton Observatory (http://xmm.vilspa.esa.es/).
- *Ssoftware systems design* (e.g., ArgoUML software design community now appearing under the banner, argouml.tigris.org).

These communities are constituted by people who identify themselves with the development of one of the four kinds of software just noted. Participants within these communities often participate in different *roles* and contribute *software representations or content* (programs, artifacts, execution scripts, code reviews, comments, etc.) to *Web sites* within each community. Administrators of these cites then serve as gatekeepers in the choices they make for what information to post, when and where within the site to post it, and whether to create a *site map* that constitutes a classification of *site and community domain content*. These people also engage in *online discussion forums* or *threaded email messages* as a regular way to both observe and contribute to discussions of topics of interest to community participants. Finally, in each of the four communities we are examining, participants choose on occasion to author and publish *technical reports or scholarly research papers* about their software development efforts, which are publicly available for subsequent examination and review. Each of these highlighted items point to the public availability of data that can be collected, analyzed, and re-represented within narrative ethnographies or computational process models (Curtis, Kellner, and Over 1992, Kling and Scacchi 1982, Mi and Scacchi 1990, Scacchi 1998,1999). Significant examples of each kind of data can be readily provided for presentation at the Workshop and in the full paper.

Comparative case study framework

The software development practices of the four communities we chose to examine can be compared and contrasted in a number of ways. In this regard, we are conducting case studies in each community. Observations and findings from each such case study can be studied, analyzed, and compared:

- As individual cases
- Within a community
- Multiple cases across two communities
- Multiple cases across all communities

These set of choices for comparison implies that a we can analyze and contrast open software development practices using as many as four different levels of analysis. This multi-level comparative analysis provides a framework for constructing models of practice/process that are both empirically grounded and increasingly general in their scope (Scacchi 1998, 1999). Thus, the comparative case study framework provides a logic that draws on the strength of capturing qualitative data that can provide a rich context for interpretation of case study data though with limited generalization. At the same time, this framework mitigates against the limits of generality assigned to an individual case study through the comparative, crosscutting, and interrelated (e.g.,

hyperlinked) analyses of multiple cases. As before, examples of each level of case data analysis can be readily provided for presentation at the Workshop and in the full paper.

Comparative case studies are important in that they can serve as foundation for the formalization of our findings and process models as a *process meta-model* (Mi and Scacchi 1996). A process meta-model is also used to configure, generate, or instantiate Web-based process modeling, prototyping, and enactment environments that enable modeled processes to be globally deployed and computationally supported (Scacchi and Noll 1997, Noll and Scacchi 1999). Subsequently, we now turn to highlight the software development processes that have been put into practice within different open software communities.

Open software development processes

In contrast to the world of academic software engineering, open software development communities do not seem to readily adopt or practice modern software engineering processes. These communities do develop software that is extremely valuable, generally reliable, widely available, and readily useable within its associated user community. So, what development processes are being routinely used and practiced?

From our studies to date, we have found five types of software development processes being employed across all four communities that merit close examination.

- Requirements analysis and specification
- Coordinated version control, system build, and staged incremental release
- Maintenance as evolutionary redevelopment, refinement, and redistribution
- Project management
- Software technology transfer

Each process will be shown to differ from traditional software engineering prescriptions, though none should be construed as independent or more important than the others should. Furthermore, it appears that these processes may occur concurrent to one another, rather than strictly or partially ordered within a traditional life cycle model. As before, examples of these processes (i.e., process instances) can be provided for presentation at the Workshop and in the full paper.

Conclusions

Open software development practices are giving rise to a new view of how complex software systems can be constructed, deployed, and evolved. Open software development does not adhere to the traditional engineering rationality found in the legacy of software engineering life cycle models or prescriptive standards. Open software development is inherently a complex web of socio-technical processes, development situations, and dynamically emerging development contexts (Kling and Scacchi 1982). This position paper presents an empirical framework that begins to outline some of the contours and dynamics that characterize how open software systems and their associated communities of practice are intertwined and mutually situated to the benefit of both.

Acknowledgements

The research described in this report is supported by a grant from the National Science Foundation #IIS-0083075 for "Understanding Open Software Communities, Processes and Practices", and from the Defense Acquisition University by grant N487650-27803. No endorsement implied. Mark Ackerman and Jack Muramatsu at the UCI Institute for Software Research are collaborators on this research described in this paper.

References

B. Curtis, M.I. Kellner and J. Over, Process modeling, *Communications ACM* 35, 9, 75 - 90, 1992.

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

J. Feller and B. Fitzgerald, A Framework Analysis of the Open Source Software Development Paradigm, *Proc. 21st. Intern. Conf. Information Systems* (ICIS), 58-69, 2000.

K. Fogel, *Open Source Development with CVS*, Coriolis Press, Scottsdale, AZ, 1999.

R. Kling and W. Scacchi, The Web of Computing: Computer technology as social organization. In M. Yovits (ed.), *Advances in Computers*, Vol. 21, 3-90. Academic Press, New York, 1982.

P. Mi and W. Scacchi, A Knowledge-based Environment for Modeling and Simulating Software Engineering Processes. *IEEE Trans. Knowledge and Data Engineering*, 2(3), 283-294, Sept 1990.

P. Mi and W. Scacchi, A Meta-Model for Formulating Knowledge-Based Models of Software Development. *Decision Support Systems*, 17(3), 313-330. 1996.

A. Mockus, R.T. Fielding, and J. Herbsleb, A Case Study of Open Software Development: The Apache Server, *Proc. 22nd. International Conf. Software Engineering*, Limerick, IR, 263-272, 2000.

J. Noll and W. Scacchi, Supporting Software Development in Virtual Enterprises. *J. Digital Information*, 1(4), February 1999.

W. Scacchi, Modeling, Simulating, and Enacting Complex Organizational Processes: A Life Cycle Approach, in M. Prietula, K. Carley, and L. Gasser (eds.), *Simulating Organizations: Computational Models of Institutions and Groups*, AAAI Press/MIT Press, Menlo Park, CA, 153-168, 1998.

W. Scacchi, Experience with Software Process Simulation and Modeling, *J. Systems and Software*, 46,183-192, 1999.

W. Scacchi and J. Noll, Process-Driven Intranets: Life-Cycle Support for Process Reengineering. *IEEE Internet Computing*, 1(5), 42-49, September-October 1997.

L. Smarr and S. Graham, Recommendations of the Panel on Open Source Software for High End Computing, Presidential Information Technology Advisory Committee (PITAC), http://www.ccic.gov/ac/pres-oss-11sep00.pdf, September 2000.

Understanding the Social, Technological, and Public Policy Implications of Open Source Software Development

Position paper

Walt Scacchi (wscacchi@uci.edu),

Institute for Software Research,

University of California, Irvine.

August 2002

Interest in open source software has emerged in many different communities. Much of this interest has focused attention primarily onto the products of open software development (source code), and secondarily onto the processes and productive units that facilitate such development. My research is focused to understanding the processes, practices, and communities that give rise to open source software. My research group is studying (a) the role of software informalisms (vs. formalisms and standards found in software engineering), (b) the emergence and articulation of open software requirements, (c) the forms and constituencies of the social worlds of open software, and (d) other processes and practices across multiple open software development communities [Scacchi 2001b, 2002a,b,c]. I am prepared to discuss our results, work in progress, and the need for further research on all of these topics. However, the remainder of this position paper identifies what I believe are areas, topics, or basic questions requiring further research in the arena of open source software development and how it may impinge on government policies. These follow in an unordered manner.

Understanding the *quality* of open source software from a socio-technical perspective

What is the best, most effective way to determine the quality of open source software products and development processes? Open source software is developed in a highly social online environment where developers are frequently dispersed in space and time, but may rarely if ever meet for face-to-face interaction. If so, how is the distributed asynchronous collaboration among developers brought into being and sustained over time? Does this mode of computer-supported collaborative work increase, decrease, or have no significant effect on the quality of the open source software? Elsewhere, it seems that the quality of open source software has been called into question, indicating that open source software can sometimes by very problematic and plagued with many critical bugs/errors. Public repositories of bug reports, errors, and other related problems/issues hosted on the Web, like Bugzilla and IssueZilla, contain hundreds to hundreds of thousands of reports for different, popular open source software systems (e.g., the Mozilla Web browser). How might these repositories be cultivated and their data systematically analyzed to see what kinds of first, second, or higher-order patterns might exist? How might such patterns better reveal the intertwined social and technical features that shape and evolve open software quality?

Understanding the occupational cultures and career contingencies of open software developers

Why people want to allocate a portion of their available time, skill, and sentiment to develop open source software is unclear. Many social constructed conditions or variables are often mentioned, including building trust and reputation; "geek fame"; being generous with one's time, expertise, and source code; and creating and contributing software as public goods or gifts [Pavlicek 2000]. However, many of the more vocal proponents of open source software [e.g., DiBona, *et al.* 2000] have financially profited or accumulated significant amounts of social capital based on the legacy of their experience and open software contributions. Do such benefits accumulate only to "early adopters" or advocates of open source software development?

Many developers of open source software do so as part of their paid job, or as a way of demonstrating their technical, communication, or social interaction skills as a way to get a (better) job. Are these conditions unique to software developers in general, or open source software developers in particular? Elsewhere, what is the role of women in the open software development community? Does open source software development, as a mode or style of technical work, tend to encourage, discourage, or have little effect in encouraging women to develop a career in the software R&D area? Last, though open source software projects engage developers who may be globally distributed, does such distribution cross all, some, or few ethnic boundaries?

Understanding the role of open source software in advancing (or inhibiting) research in the natural and physical sciences

It appears in the U.S. that many Federal research agencies will face the issue of whether or not to adopt a policy that mandates or encourages that all software developed with agency funding must be open source. However, what does such a policy imply for the advancement of scientific research and development in disciplines inside or outside of the computing research community? For example, the development of the national computation grid [Foster *et al.,* 2001] seems to rely on the use of open source software. But this community involves researchers with long track records in computer science or computational science research. In the Astrophysics community, there is growing enthusiasm for research and development of a computational "astrogrid" that can integrate dispersed astrophysical sensors and software systems to act as a *national virtual observatory* [NVO 2002]. NASA and the National Science Foundation are principal agencies funding such a grid. What principles, guidelines, or best practices should guide the development of a national research infrastructure like a virtual observatory that is to be built by astrophysicists using open source software development techniques? Elsewhere, the international medical informatics, bioinformatics (cf. http://bioinformatics.org/about and http://open-bio.org/bosc2002/), and ecological systems (http://www.open-research.org/) communities have already begun discussing why future medical, biological, and ecological research systems should be open source. However, the genomics and proteinomics communities seem reluctant to embrace open source systems or open databases, since there is still a great rush of patents being filed pertaining to intellectual property of a microbiological origin that can primarily be accessed and manipulated via software-intensive systems and instrumentation. Should a separate pool of Federal biotechnology research investments be targeted to those who develop open source

software or open databases as part of their research (cf. http://www.openinformatics.org/petition.html)?

Understanding the role of open source software in national and international science policy

What could we learn from by comparing and contrasting efforts in the U.S., European Union, and High-Tech Asia (Japan, Korea, Taiwan, Singapore), as well as to other parts of the World in the encouragement of open source software development? For example, the European Commission's Information Society Technologies Programme [IST 2002], currently budgeted at 3.6B Euros over five years, stipulates all software developed with programme funding *should* be open source whenever possible. Does the IST programme represent a competitive advantage, disadvantage, or simply an alternative compared to U.S. Federal R&D investment which does not require open source software to be developed with its funds? Will it advantage or disadvantage the U.S. software industry, only certain firms in that industry (e.g., Microsoft, Oracle, Sun, or Adobe), or have little/no consequence? Will the IST programme policy towards open source software give rise to new products or services that are superior, inferior, or just different than those arising from U.S. funded R&D? Is a "hands-off" open source policy (i.e., the current U.S. position) or a "hands-on" policy better or worse for High-Tech Asia economies, or for emerging nation economies?

We do note that the President's Information Technology Advisory Committee [2000] report strongly advocates that all high-end computing software funding by the U.S. research agencies adopt a hands-on, open source technology policy. Should the U.S. research agencies follow the PITAC recommendation, and align or distance themselves with the policy objectives of the IST programme? Would such an alignment or distancing help, hurt, neutralize monopoly (or market dominance) positions of the U.S. software industry, or certain firms within it?

The emergence of "open government", arising from the integration of concepts and practices from open source and E-Government

How can open source development concepts be brought to bear in the realm of digital government? Digital government (or E-Government) encourages the adoption of modern IT business practices that exploit the World-Wide Web and Electronic Commerce capabilities to improve the government operations and public services. NSF's Digital Government initiative supports research in IT security and privacy [DG.O 2002], as well as the collection, statistical analysis, public access, and visual display of very large data bases of public data [DGRC 2002]. Other parts of the U.S. government are investigating E-Government strategies for procurement and acquisition [Scacchi and Boehm 1998, Scacchi 2001a], data storage and data entry (e.g., electronic filings of tax forms by individuals, and SEC forms by businesses), E-catalog based retail product sales (U.S. Mint), and smart cards [Steyaert 2001]. Outside of the U.S., nearly a dozen countries are already formulating legislation that requires or expresses preference for the adoption and use of open source software systems by national or regional government agencies [Anonymous 2002, Cortiana 2002, Nunez 2002].

In contrast, *open government* seeks to open for public sharing, discussion, review, ongoing development and refinement, and unrestricted reproduction (replication and redistribution) the "source code" of the products and processes of the business of government. Open government represents a concept that seeks more than just the adoption and use of open source software systems by government agencies. This concept seeks to explore the potential and opportunities that can emerge when one views the purpose of digital government as also including how to empower and engage an interested public in better understanding how government processes and practices can be made better, cheaper, and faster through the development of open source processes, practices, and communities of practice for government operations. Would open government allow for the establishment of community Web portals or other open testbeds (e.g., a national virtual government observatory or computational grid) where alternative government processes or practices might be (re)designed, prototyped, and evaluated via collaborative experimentation and engagement [Scacchi and Boehm 1998, Scacchi 2001a, 2002c]? Would open government systems provide new modes of access and participation in an open democracy through the development, use, and collaborative evolution by interested government system developers, industry, and citizens? Would open government represent a new avenue to explore how government operations might be made accessible for educational purposes in high school (grades 9-12) and college settings? Would open government enable more complete assessment of the financial and infrastructural costs/benefits of new legislation that is created and imposed, but otherwise be unfunded?

Overall research needs

The research community needs a better articulation and understanding of "critical mass" issues in open source software development projects. Are characteristics of large software projects working on "Internet time" [Cusumano and Yoffie 1999, MacCormack, Verganti and Iansiti 2001] fundamentally different than those projects which lack critical mass features? What may work well or be true about specific open source software development projects like the GNU/Linux operating system, Apache Web server, Mozilla Web browser, SendMail, and BIND, may not necessarily be indicative of the characteristics and critical success/failure factors of other open source software projects. For example, the SourceForge Web portal (http://www.sourceforge.net) currently hosts more than 49K open source software projects (with 70 new projects added per day) and more than 450K registered users (with 700 new users registering per day). None of the major open source software projects like GNU/Linux, Apache, or Mozilla are found on SourceForge or Freshmeat.org. Furthermore, of the 49K projects at least 10% are identified by their developers as *production quality and stable*, thus suitable for routine use by end-users who primarily want to use, rather than develop, such software. Similarly, at least 10% of the total set of projects is no further than the *interesting idea* stage of development. Many of these projects will wither due to an inability to realize their effort as a successful open source community motivated to develop and sustain an open software system. Thus, studies indicative of a single open source software project, or multiple projects of a similar kind, may produce results that cannot be generalized to other/smaller open source projects. Similarly, the results of such myopic studies may lead or mis-lead new open source projects in R&D communities with little prior experience with open source software products, processes, productive units, or information infrastructures. Thus, premature generalization of the desirable

features or characteristics of open source projects like GNU/Linux and Apache Web Server, or their adoption be particular government agencies, may give rise to inappropriate conclusions about what the best practices of open source software development really are for projects that lack critical mass and Internet time characteristics.

The scientific and policy research communities need to encourage comparative empirical study of open source software development products, processes, productive units, and information infrastructures that span different R&D communities and government agencies within and across nations. This is especially true for communities whose research or system usage interests are not primarily rooted in software system development (e.g., medicine, astrophysics, genomics/proteinomics, environmental ecology, national research ministries, and other government agencies) but who may believe that open source software is a better or best way to develop their software systems. We need to better understand how communities are similar or different in their patterns of software system development and use, and how open source processes and practices align within these communities and patterns.

Last, the scientific and policy research community needs to encourage interdisciplinary study of open source software development, especially studies rooted in understanding and advancing economic analyses, innovation theory, and collaborative software development processes and practices.

Acknowledgements

The research described in this report is supported by grants from the National Science Foundation #IIS-0083075, the Defense Acquisition University by contract N487650-27803, and from the NSF Industry-University Partnership in the CRITO Consortium at UCI. No endorsement implied. Mark Ackerman at the University of Michigan, as well as Mark Bergman, Margaret Elliott, and Xiaobin Li at the UCI Institute for Software Research, are collaborators on the research described in this paper.

References

(Anonymous), Legislation on the use of Free Software in the Government, http://www.lugcos.org.ar/serv/mirrors/proposicion/doc/referencias/#ref.#1, August 2002.

F. Cortiana, "Norms in matter of computer science pluralismo on the adoption and the spread of the free software and on the portabilità of documents informed to us in Public Administration" (XIV Legislatura Action Senate n. 1188) (in Italian) April 2002. http://www.softwarelibero.it/altri/cortiana.shtml (English translation, http://translate.google.com/translate?hl=en&sl=it&u=http://www.softwarelibero.it/altri/cortiana.shtml&prev=/search%3Fq%3Dwww.softwarelibero.it/altri/cortiana.shtml%26num%3D100%26hl%3Den%26lr%3D%26ie%3DUTF-8)

M. Cusumano and D.B. Yoffie, Software Development on Internet Time, *Computer*, 32(10), 60-69, October 1999.

C. DiBona, S. Ockman and M. Stone, *Open Sources: Voices from the Open Source Revolution*,

O'Reilly Press, Sebastopol, CA, 1999.

(DG.O) Digital Government.Org, http://www.diggov.org, 2002.

(DGRC) Digitial Government Research Center, http://www.isi.edu/dgrc/, 2002.

FastLane, National Science Foundation, https://www.fastlane.nsf.gov/, 2002.

I. Foster, C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *Intern. J. High Performance Computing Applications*, 15(3), 200-222, 2001.

(IST) Information Society Technologies Programme, The European Commission, http://www.cordis.lu/ist/, 2002.

A.MacCormack, R. Verganti, and M. Iansiti, Developing Products on Internet Time: The Anatomy of a Flexible Development Process, *Management Science*, 47(1), January 2001.

(NVO) National Virtual Observatory, http://www.us-vo.org/, 2002.

E.D.V. Nunez, *Peruvian Congressman Refutes Microsoft "Fear, Uncertainty, and Doubt" (F.U.D.) Concerning Free and Open Source Software*, 8 April 2002, (original in Spanish), http://www.opensource.org/docs/peru_and_ms.html.

R. Pavlicek, *Embracing Insanity: Open Source Software Development*, SAMS Publishing, Indianapolis, IN, 2000.

(PITAC) President's Information Technology Advisory Committee, *Developing Open Source Software to Advance High-End Computing*, September 2000. http://www.ccic.gov/pubs/pitac/pres-oss-11sep00.pdf

W. Scacchi, Redesigning Contracted Service Procurement for Internet-based Electronic Commerce: A Case Study, *J. Information Technology and Management*, 2(3), 313-334, 2001a.

W. Scacchi, Software Development Practices in Open Software Development Communities: A Comparative Case Study, presented at The 1st Workshop on Open Source Software Engineering, Toronto, Ontario, May 2001b.

W. Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings - Software*, 149(1), 24-39, 2002a.

W. Scacchi, Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering?, *Proc. 2nd. Workshop on Open Source Software Engineering*, Orlando, FL, May 2002b.

W. Scacchi, Exploring Open Software System Acquisition Processes and Architectures, Final Report, Institute for Software Research, University of California, Irvine, July 2002c.

W. Scacchi and B.E. Boehm, Virtual System Acquisition: Approach and Transitions, *Acquisition Review Quarterly*, 5(2), 185-216, Spring 1998. Also see,

http://www.dsmc.dsm.mil/pubs/arq/98arq/scacchi.pdf

J. Steyaert, (Deputy associate administrator, Office of Information Technology, GSA), View From the Top: What Government is Doing, *Solution Series Conf. On E-Government*, 24 April 2001. Webcast version at http://www.gcn.com/webcast/070201gcn.html

IEE Proceedings -- Software
Paper number 29840, Accepted for publication with revisions, December 2001.

Understanding the Requirements for Developing Open Source Software Systems

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425 USA
http://www.ics.uci.edu/~wscacchi
wscacchi@ics.uci.edu

Abstract

This study presents an initial set of findings from an empirical study of social processes, technical system configurations, organizational contexts, and interrelationships that give rise to open software. The focus is directed at understanding the requirements for open software development efforts, and how the development of these requirements differs from those traditional to software engineering and requirements engineering. Four open software development communities are described, examined, and compared to help discover what these differences may be. Eight kinds of software informalisms are found to play a critical role in the elicitation, analysis, specification, validation, and management of requirements for developing open software systems. Subsequently, understanding the roles these software informalisms take in a new formulation of the requirements development process for open source software is the focus of this study. This focus enables considering a reformulation of the requirements engineering process and its associated artifacts or (in)formalisms to better account for the requirements for developing open source software systems.

Revised version appears in:

***IEE Proceedings--Software*, 149(1), 24-39, February 2002.**

1. Overview

The focus in this paper is directed at understanding the requirements for open software development efforts, and how the development of these requirements differs from those traditional to software engineering and requirements engineering [10, 17, 22, 28]. It is not about hypothesis testing or testing the viability of a prescriptive software engineering methodology or notational form. Instead, this study is about discovery, description, and abstraction of open software development practices and artifacts in different settings in different communities. It is about expanding our notions of what requirements engineering processes and process models need to address to account for open source software development. But to set the stage for such an analysis, we first need to characterize the research methods and principles employed in this study. Subsequently, these are used to understand what open software communities are being examined, and what characteristics distinguish one community from another. This is followed by the model of the processes and artifacts that give rise to the requirements for developing open source software systems. The model and artifacts are the focus of the analysis and basis of the concluding discussion. This includes a discussion of what is new or different in the findings presented in this report, as well as some of their implications for what can or should be formalized when developing different kinds of open software systems.

2. Research methodology: comparative methods and principles

This study reports on findings and results from an ongoing investigation of the socio-technical processes, work practices, and community forms found in open source software development. The purpose of this multi-year investigation is to develop narrative, semi-structured (i.e., hypertextual), and formal computational models of these processes, practices, and community forms. This report presents a systematic narrative model that characterizes the processes through which the requirements for open source software systems are developed. The model compares in form, and presents a contrasting account of, how software requirements differ between traditional software engineering and open source approaches. The model is descriptive and empirically grounded. The model is also comparative in that it attempts to characterize an open source requirements engineering process that transcends the practice in a particular project, or within a particular community. This comparative dimension is necessary to avoid premature generalizations about processes or practices associated with a particular open software system or those that receive substantial attention in the news media (e.g., the GNU/Linux operating system). Such comparison also allows for system projects that may follow a different form or version of open source software development (e.g., those in the Astrophysics research community or networked computer game arena). Subsequently, the model is neither prescriptive nor proscriptive in that it does not characterize what should be or what might be done in order to develop open source software requirements, except in the concluding discussion, where such remarks are bracketed and qualified.

Comparative case studies are also important in that they can serve as foundation for the formalization of our findings and process models as a *process meta-model* [24]. Such a meta-model can be used to construct a predictive, testable, and incrementally refined theory of open software development processes within or across communities or projects. A process meta-model can also be used to configure, generate, or instantiate Web-based process modeling, prototyping, and enactment environments that enable modeled processes to be globally deployed and computationally supported [e.g., 26, 27]. This may be of value to other academic research or commercial development organizations that seek to adopt "best practices" for open software development processes well suited to their needs and situation. Therefore, the study and results presented in this report denote a new foundation on which computational models of open software requirements processes may be developed, as well as their subsequent analysis, simulation, or redesign [34].

The study reported here entails the use of empirical field study methods [40] that follow conform to the principles for conducting and evaluating interpretive research design [19] as identified here. Seven principles are used in this study in the following manner.

The first principle is that of the *hermeneutic circle,* here focusing attention on the analysis of the whole open source requirements process, how it emerges from consideration of its parts (i.e., requirements sub-processes and associated artifacts), and how the whole interacts with and shapes these parts.

The second principle is that of *contextualization,* which draws attention to the need to identify a web of situations, conditions, or events that characterize the social and historical background of requirements engineering practices found in open source development projects or communities [20]. This begins in Sections 3 and 4, then continues throughout the presentation of the descriptive model and informal artifacts of open software requirements processes.

The next principle is that of *revealing the interaction of the researcher and the subjects/artifacts*. This is a basic concern that must be addressed and disclosed whenever the research involves an ethnographic field study, particularly those requiring virtual ethnography [16], as is the situation here. In this study, the researcher acted as a participant observer who seeks to understand how open source software requirements for a set of specific open software systems are developed, and to what ends. In the virtual worlds of open software development projects, there is generally no corporate workplace or single location where software development work occurs. Thus, traditional face-to-face ethnography and visible participant observation cannot readily occur. What occurs, and where it occurs, is generally online (i.e., hosted on a Web site or interactively accessed via Internet mechanisms), open to public access[1], and dispersed across geographic space and multiple time zones. Informal hallway conversations, as well as organized and scheduled meetings (rare though they may be), generally take place in an electronic and publicly visible online manner, though the requirements development work itself may be implied, hidden, or otherwise invisible. Subsequently, as a Web-based participant, the researcher could "sit in" or lurk on a group chat among core developers when it was a pre-announced event, as casual developers or other reviewers regularly do.

Alternatively, like many others potential or active participants, one could simply browse email, community bulletin boards (bboards), and related Web site postings that signal the occurrence of events or situations of interest (e.g., software release announcements or problem reports). These modes of participation are not uncommon, and are cited as one way how project newcomers can join in and learn the domain language and issues, with minimal bother or distraction of those doing the core development effort [11, 29, 32]. Social interaction among open software project participants may rarely, if ever, be face-to-face or co-located in most open source development efforts. However, real-world events like professional conferences may enable distant collaborators to meet, interact with, and learn about one another, but such events may occur only once a year, or be effectively inaccessible to project participants due to its distant location.

In open software projects, social and technical interaction primarily occurs in a networked mediated computing environment populated with Web browsers, email/bboards (and sometimes instant messaging) utilities, source text editors, and other software development tools (e.g., compilers, debuggers, and version control systems [14]). Each program/tool runs asynchronously in different end-user processes (application windows) appearing on a graphic user interface, as well as appearing as artifacts stored and distributed across one or more repositories [26, 27]. The workplace of open source software development is on the screen together with the furniture and surroundings that house it. This workplace is experienced, navigated, and interacted through keystrokes and mouse/cursor movement gestures in concert with what is seen, read, or written (typed). Thus, to observe, participate, and comprehend what's going on in an open source development project, it is necessary to become situated, embedded, and immersed as a software contributor, reviewer, discussant, or reader within such projects, and within such a networked computing environment and workplace setting.

In all the projects reported here, the researcher was a reader who acted as an interested but unfamiliar software developer seeking to understand, review, or discuss with other participants contemporary or legacy software development problems, opportunities, features, and constraints here [cf. 20]. This occurred while asynchronously running the end-user application sessions and networked computing environment indicated here, for between 0.5-10+ hours at a time, totaling more than 200 hours over a period of 10 months. The data exhibits that appear in the Section 5 are a comparative though minute sample of such

[1] Some modes of participation may be restricted--for example, anyone may check out and download source code, but typically only those approved by the core development team may upload and check in modified code into a shared repository [14].

Web-based experiences and shared artifacts, all presented as screen displays as could be seen by a community participant, as captured while and where they encountered by the researcher.

The fourth principle is that of *abstraction and generalization*. The choice to examine and compare requirements engineering activities and artifacts across four different software development communities is the response to this motivation. The requirements engineering process in Section 5 provides both a description and comparison that spans four distinct communities. The model abstracts details that are presented as summary terms (identified by sub-section headings) that span multiple open source projects in the sample space in order to create a more general model that covers the details across all the examined projects. Similarly, the classification of open source software requirements artifacts as software informalisms in Section 6 also reflects a similar kind of generalization across project and across community.

The fifth principle is that of *dialogical reasoning* which compares the received wisdom of extant theory or methodology in the software requirements engineering community, with that found empirically through participant observation in open source software development efforts. This appears in Sections 4 and 5.

The sixth principle is that of *multiple interpretations* which highlights the need to recognize that different participants see and experience things differently, though they may still be able to communicate and share these things with some degree of similarity or replication. The descriptive model presented in this report is a unique characterization that does not appear in any of the open source software development efforts or communities described. Thus, the interpretation here is therefore subject to the seventh and last principle, which is that of *suspicion to possible biases or systematic distortions* in this presentation. The reader is cautioned to look for alternative explanations or arrangements of data that might give rise to a different model of the requirements process to that which follows. If found, such models should be published as a contribution for review and comparison to the one presented here. Alternatively, the model presented here could be revised and updated to account for alternative interpretations, as a further generalization that better accounts for the other principles listed here. These issues are addressed in Sections 7 and 8.

3. Understanding open software development across different communities

We assume there is no *a priori* model or globally accepted framework that defines how open software is or should be developed. Subsequently, our starting point is to investigate open software practices in different communities from an ethnographic perspective [2, 28, 38].

We have chosen four different communities to study. These are those centered about the development of software for networked computer games, Internet/Web infrastructure, X-ray astronomy and deep space imaging, and academic software design research. In contrast to efforts that draw attention to generally one (but sometimes many) open source development project(s) within a single community [e.g., 11, 32], there is something to be gained by examining and comparing the communities, processes, and practices of open software development in different communities. This may help clarify what observations may be specific to a given community (e.g., GNU/Linux projects), compared to those that span multiple, and mostly distinct communities. In this study, two of the communities are primarily oriented to develop software that supports scholarly research (X-ray astronomy and academic software design research) with rather small user communities. In contrast, the other two communities are oriented primarily towards software development efforts that may replace/create commercially viable systems that are used by large end-user communities. Thus, there is a sample space that allows comparison of different kinds. So to begin, each community in this study can be briefly characterized.

3.1 Networked computer game worlds

Participants in this community focus on the development and evolution of first person shooters (FPS) games (e.g., *Quake Arena*, *Unreal Tournament*), massive multiplayer online role-playing games (e.g., *Everquest*, *Ultima Online*), and others (e.g., *The Sims* (maxis.com), *Neverwinter Nights* (bioware.com)). Interest in networked computer games and gaming environments, as well as their single-user counterparts, has exploded in recent years as a major (now global) mode of entertainment and playful fun. The release of *DOOM*, an early FPS game, onto the Web in open source form in the mid 1990's, began what is widely recognized the landmark event that launched the development and redistribution of so-called PC game

mods [Cleveland 2001]. Mods are updates to or variants of proprietary (closed source) computer game engines that provide extension mechanisms like game scripting languages to modify and extend a game. Mods are created by small numbers of users who want and are able to modify games, compared to the huge numbers of players that enthusiastically use the games as provided. The scope of mods has expanded to include new game types, game character models and skins (surface textures), levels (game play arenas or virtual worlds), and artificially intelligent game bots (in-game opponents). The companies, id Software (makers of *DOOM* and the *Quake* game family) and Epic Games (makers of the *Unreal* game family) helped encourage the open extension of proprietary game engines. This was done through both game licenses that require publicly distributed mods to be open source, and the provision of mod tools (level editors, model builders) and game engine programming or scripting languages for modifying game objects, behavior, as well as the potential to create entirely new games.[2] These companies also recruit new game development staff from the community of mod developers (see Exhibit 4).

3.2 Internet/Web infrastructure

Participants in this community focus on the development and evolution of systems like the Apache web server, Mozilla Web browser[3], K Development Environment (KDE), InterNet News server, OpenBSD, `mono` (an open source implementation of .NET, mostly independent from Microsoft), and thousands of others[4]. This community can be viewed as the one most typically considered in popular accounts of open source software projects. The GNU/Linux operating system environment is of course the largest, most complex, and most diverse sub-community within this arena, so much so that it merits separate treatment and examination. Many other Internet or Web infrastructure projects constitute recognizable communities or sub-communities of practice. The software systems that are the focus generally are not standalone end-user applications, but are often targeted at *system administrators* or *developers as the targeted user base,* rather than the eventual end-users of such systems. However, notable exceptions like Web browsers, news readers, instant messaging, and graphic image manipulation programs are growing in number within the end-user community

3.3 X-ray astronomy and deep space imaging

Participants in this community focus on the development and evolution of software systems supporting the Chandra X-Ray Observatory, the European Space Agency's XMM-Newton Observatory, the Sloan Digital Sky Survey, and others. These are three highly visible astrophysics research projects whose scientific discoveries depend on processing remotely sensed data through a complex network of open source software applications that process remotely sensed data [35]. In contrast to the preceding two development oriented communities, open software can play a significant role in scientific research communities. For example, when scientific findings or discoveries resulting from remotely sensed observations are reported[5], then members of the relevant scientific community want to be assured that the results are not the byproduct of

---

[2] *Unreal* begat *Half-Life* under a proprietary license, which gave rise to *Half-Life: CounterStrike*, the most popular FPS game (and game mod) at present [7]. *Counter-Strike* was developed and distributed as open source by two independent game player-developers. These developers were then financially engaged by Half-Life's commercial developer, Valve Software, to participate in the royalty stream generated from retail sales of *CounterStrike*, though the CS mod is still publicly accessible on the Web.

[3] It is reasonable to note that the two main software systems that enabled the World Wide Web, the NCSA Mosaic Web browser (and its descendants, like Netscape Navigator and Mozilla), and the Apache Web server (originally know as "HTTPd") were originally and still remain active open source software development projects.

[4] The SourceForge community web portal (http://www.sourceforge.net) currently stores information on more than 250K developers and 30K open source software development projects, with more than 10% of those projects indicating the availability of a mature, released, and actively supported software system.

[5] For example, see http://antwrp.gsfc.nasa.gov/apod/ap010725.html which displays a composite image constructed from both X-ray (Chandra Observatory) and optical (Hubble Space Telescope) sensors. The open software processing pipelines for each sensor are mostly distinct and are maintained by different organizations. However, their outputs must be integrated, and the images must be registered and oriented for synchronized overlay, pseudo-colored, and then composed into a final image, as shown on the cited Web page. There are dozens of open software programs that must be brought into alignment for such an image to be produced, and for such a scientific discovery to be claimed and substantiated [31, 35].

some questionable software calculation or opaque processing trick. In scientific fields like astrophysics that critically depend on software, open source is increasingly considered an essential precondition for research to proceed, and for scientific findings to be trusted and open to independent review and validation. Furthermore, as discoveries in the physics of deep space are made, this in turn often leads to modification or extension of the astronomical software in use in order to further explore and analyze newly observed phenomena, or to modify/add capabilities to how the remote sensing mechanisms operate.

3.4 Academic software systems design

Participants in this community focus on the development and evolution of software architecture and UML centered design efforts, such as for ArgoUML (http://argouml.tigris.org) or xARCH at CMU and UCI (http://www.isr.uci.edu/projects/xarch/). This community can easily be associated with a mainstream of software engineering research. People who participate in this community generally develop software for academic research or teaching purposes in order to explore topics like software design, software architecture, software design modeling notations, software design recovery (reverse software engineering), etc. Accordingly, it may not be unreasonable to expect that open software developed in this community should embody or demonstrate principles from modern software engineering theory or practice. Furthermore, much like the X-ray astronomy community, members of this community expect that when breakthrough technologies or innovations have been declared, such as in a refereed conference paper or publication in a scholarly journal, the opportunity exists for other community members to be able to access, review, or try out the software to assess and demonstrate its capabilities. An inability to provide such access may result in the software being labeled as "vaporware" and the innovation claim challenged, discounted, or rebuked. Alternatively, declarations of "non-disclosure" or "proprietary intellectual property" are generally not made for academic software, unless or until it is transferred to a commercial firm. However, it is often acceptable to find that academic software, whether open source or not, constitutes nothing more than a "proof of concept" demonstration or prototype system, not intended for routine or production use by end-users.

3.5 Community Characteristics

Each community is constituted by people who *identify themselves* with the development of open software within one of the four arenas noted above.[6] Though participants may employ pseudonyms (user-id's) in identifying themselves within a community, they do not assume nor rely on anonymous identifiers, as is found in other communities for socializing in cyberspace [Preece 2000, Smith and Kollock 1999]. Open software developers or contributors tend to act in ways where *building trust and reputation*, "*geek fame*", and *being generous with one's time, expertise, and source code* are valued traits of community participants [Pavlicek 2000]. They work to develop and contribute *software representations or content* (programs, design diagrams, execution scripts, code reviews, test case data, Web pages, email comments, etc.) to *Web sites* within each community. *Making contributions*, and *being recognized by other community members as having made substantive contributions*, is often a prerequisite for advancing technically and socially within a community [Fielding 1999, Kim 2000]. As a consequence, participants within these communities often participate in different *roles* within both *technical* and *social networks* [Smith and Kollock 1999, Preece 2000] in the course of developing, using, and evolving open software.

Administrators of open software community Web sites 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 [36]. Similarly, they may choose to create a *site map* that constitutes a classification of *site and domain content,* as well as a geography of *community structure and boundaries* [4]. Community participants regularly use bboards to 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 [39]. 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. Finally, in each of the four communities examined here, participants choose on occasion to author and publish *technical reports or scholarly research papers* about their software development efforts, which are publicly available for subsequent examination, review, and secondary analysis.

[6] An alternative scheme for automatically discovering the membership of a Web-based community uses graph traversal (crawling linked web pages) and s-t maximum flow network algorithms [13].

Each of these highlighted items point to the public availability of data that can be collected, analyzed, and re-represented within narrative ethnographies [16, 20], computational process models [8, 24, 27, 34], or for quantitative studies [6, 21]. Significant examples of each kind of data have been collected and analyzed as part of this ongoing study. This paper includes a number of examples that serve as this data. Subsequently, we turn to review what requirements engineering is about, in order to establish a baseline of comparison for whether what we observe with the development of open software system requirements is similar or different, and if so how.

4. The classic software requirements engineering process

Experts in the field of software requirements engineering identify a recurring set of activities that characterize this engineering process [10, 17, 22, 28]. These activities, which are generally construed as necessary in order to produce a reliable, high quality, trustworthy system, include:

Eliciting requirements: identifies system stakeholders, stakeholder goals, needs, and expectations, and system boundaries. Elicitation techniques like questionnaire surveys, interviews, documentation review, focus groups, or joint application development (JAD) team meetings may be employed.

Modeling or specifying requirements: focuses attention to the systematic modeling of both functional and non-functional software requirements. One can model functional requirements of operational domain problems by specifying system processing states, events (input events, output events, process execution flags or signals, error detection, and exception handling triggers), and system data. Specifying system data may include identification of data objects, data types, data sources, end-user screen displays, and meta-data, as well as construction of a data dictionary. Functional requirements should also specify system data flow through system or subsystem states as controlled or synchronized by events. Beyond this, advanced modeling techniques may include construction of visual animations or simulated walkthroughs of overall system functionality. In contrast, one can model non-functional requirements as goals, capabilities, and constraints that situate the functional system within some context of operation. This can involve identifying an enterprise model, problem domain model, system model type, and data model type.

Analyzing requirements: entails a systematic reasoning of the internal consistency, completeness, or correctness of a specification. It does not check to see if the requirements are externally correct or an accurate model of the world. That determination may result from observing a visual animation of the specification during operational execution (a simulation). More sophisticated analyses may check for reachability, termination, live-lock and dead-lock, and safety in the modeled system.

Validating requirements: engages domain experts to assess feasibility of modeled system solution, as well as to identify realizable, plausible, and implausible system requirements. Systematic techniques for inspecting requirements to assess system usability and feasibility may also be employed. As a result of validation, the requirements engineer can better calibrate customer expectations about what can be developed.

Communicating requirements: entails documenting requirements, for example, through the creation of an software requirements specification (SRS) document, establishing criteria for requirements traceability, and managing the storage and evolution of the preceding requirements artifacts.

With these activities at hand, it is possible to consider whether this requirements engineering process captures or suitably characterizes what occurs in the development of requirements for open software systems, and how it occurs. The objective is not to establish conformity or distance metrics, nor some other quantitative measures that purport to reveal insights about the comparative quality of different open software systems, which might then be reused in other experimental situations [40]. Instead, the objective is to help determine whether the classic process adequately characterizes and covers what is observable in the development of open software requirements, or whether a new alternative model of requirements development for open software is needed.

Recently, experts in requirements engineering have begun to recognize the limits of the traditional requirements engineering process [28, 38]. They call for better modeling and analysis of the problem domain, as opposed to just focusing on the functional behavior of the software. They draw attention to the need to develop richer models for capturing and analyzing non-functional requirements. They also point to opportunities to bridge the gap between requirements elicitation techniques based on contextual and ethnographic techniques [15, 16, 38], and those techniques for formal specification and analysis. Thus, we can take into account these suggested improvements as well in our study.

5. Open software processes for developing requirements

In contrast to the world of classic software engineering, open software development communities do not seem to readily adopt or practice modern software engineering or requirements engineering processes. Perhaps this is no surprise. However, these communities do develop software that is extremely valuable, generally reliable, often trustworthy, and readily used within its associated user community. So, what processes or practices are being used to develop the requirements for open software systems?

From our study to date, we have found many types of software requirements activities being employed within or across the four communities. However, we have yet to find examples of formal requirements elicitation, analysis, and specification activity of the kind suggested by software requirements engineering textbooks [10, 22] in any of the four communities under study. Similarly, we have only found one example[7] online (in the Web sites) or offline (in published technical reports) of documents identified as "requirements specification" documents within these communities. However, what we have found is different.

5.1 Requirements elicitation vs. assertion of open software requirements

It appears that open software requirements are articulated in a number of ways that are ultimately expressed, represented, or depicted on the Web. On closer examination, requirements for open software can appear or be implied within an email message or within a discussion thread that is captured and/or posted on a project's Web site bboard for open review, elaboration, refutation, or refinement. Consider the following example found on the Web site for the KDE system (http://www.kde.org/), within the Internet/Web Infrastructure community. This example displayed in Exhibit 1[8] reveals *asserted capabilities* for the Qt3 subsystem within KDE. These capabilities (identified in the exhibit as the "Re: Benefits of Qt3?" discussion thread) highlight implied requirements for multi-language character sets (Arabic and Hebrew, as well as English), database support ("…there is often need to access data from a database and display it in a GUI, or vice versa…"), and others. These requirements are simply asserted without reference to other documents, sources, standards, or JAD focus groups--they are requirements because some developers wanted these capabilities.

Asserted system capabilities are *post-hoc* requirements characterizing a functional capability that has already been implemented. The concerned developers justify their requirements through their provision of the required coding effort to make these capabilities operational. Senior members or core developers in the community then vote or agree through discussion to include the asserted capability into the system's distribution [12]. The historical record may be there, within the email or bboard discussion archive, to document who required what, where, when, why, and how. However, once asserted, there is generally no

[7] "Software Requirements Specifications for the Central Manager (DDNS_CM) CSCI of the Distributed Data Network System (DDNS)", 29 July 2001, http://www.sourceforge.net/projects/ddns. It seems fair to observe that this SRS seems to follow guidelines embodied in military standards for software development, perhaps suggesting its origination in an industrial firm.

[8] Each exhibit appears as a screenshot of a Web browsing session. It includes contextual information, following the second research principle, thus requiring and benefiting from a more complete display view.

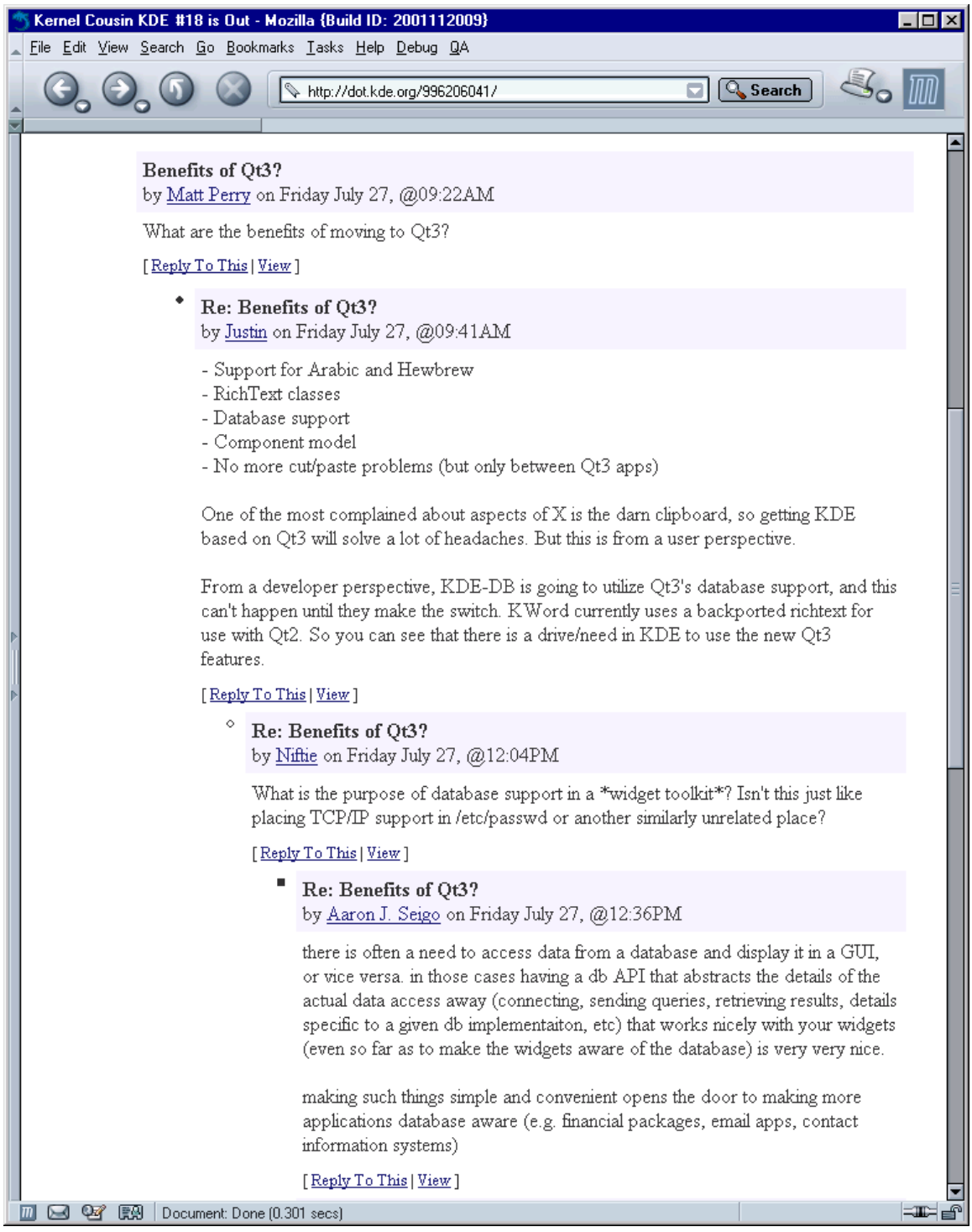


Exhibit 1. A sample of implicit requirements for the KDE software subsystem Qt3 expressed in a threaded email discussion. Source: http://dot.kde.org/996206041/, July 2001.

further effort apparent to document, formalize, or substantiate such a capability as a system requirement. Asserted capabilities then become invisible or transparent, taken-for-granted requirements that can be labeled or treated as *obvious* (i.e., a shared awareness) to those familiar with the system's development.

Another example reveals a different kind open software requirement. This case, displayed in Exhibit 2,[9] finds a requirements "vision" document that conveys a non-functional requirement for "community software development" in the bottom portion of the exhibit. This can be read as a non-functional requirement for the system's developers to embrace community software development as the process to develop and evolve the ArgoUML system, rather than say through a process which relies on the use of system models represented as UML diagrams [cf. 38]. Perhaps community software development, and by extension, community development, are recognized as being important to the development and success of this system. It may also be a method for improving system quality and reliability when compared to existing software engineering tools and techniques (i.e., those based on UML, or supporting UML-based software design).

A third example reveals yet another kind of elicitation found in the Internet/Web infrastructure community. In Exhibit 3, we see an overview of the `mono` project. Here we see multiple statements for would-be software component/class owners to sign-up and commit to developing the required ideas, run-time, (object service) classes, and projects. These are non-functional requirements for people to volunteer to participate in community software development, in a manner perhaps compatible with that portrayed in Exhibit 2. The systems in Exhibits 2 and 3 must also be considered early in their overall development or maturity, since they call for functional capabilities that are needed to help make sufficiently complete for usage.

Thus, in understanding how the requirements of open software systems are elicited, we find evidence for elicitation of volunteers to come forward to participate in community software development. A similar example inviting new participants into the world of game mods appears in Exhibit 4. We also observe the assertion of requirements that simply appear to exist without question or without trace to a point of origination, rather than somehow being elicited from stakeholders, customers, or prospective end-users of open software systems. As previously noted, we have not yet found evidence or data to indicate the occurrence or documentation of a requirements elicitation effort arising in an open software development project. However, finding such evidence would not invalidate the other observations; instead, it would point to a need to broaden the scope of how software requirements are captured or recorded.

5.2 Requirements analysis vs. requirements reading, sense-making, and accountability

In open software development, how does requirements analysis occur, and where and how are requirements specifications described? Though requirements analysis and specification are interrelated activities, rather than distinct stages, we first consider examining how open software requirements are analyzed. Exhibits 5 and 6 come from different points in the same source document, a single research paper accessible on the Web, associated with the Chandra X-ray Center Data System (CXCDS) for sensing and imaging deep space (astronomical) objects that radiate in the X-ray spectrum. Exhibit 5 suggests to the reader that the requirements for the CXCDS are involved and complex (as seen in the "Abstract"), and Exhibit 6 seems to confirm this claim, as least to an outsider interpreting Figure 2 shown in the exhibit. As a data-flow diagram, Exhibit 6 either suggests or denotes part of the specification of requirements for the CXCDS. But how do software developers in this community (astrophysicists) understand what's involved in the functional operation of a complex system like this? One answer lies in the observation that developers who seek such an understanding must read this research paper quite closely, as well as being able to draw on their prior knowledge and experience in the relevant physical, telemetric, digital, and software domains. A close reading likely means one that entails multiple re-readings and sense-making relative to one's expertise and prior development experience [cf. 1]. A more casual though competent reading requires some degree of confidence and trust in the authors' account of how the functionality of the

[9] The ArgoUML tool [33] within the academic software design community at http://argouml.tigris.org/.

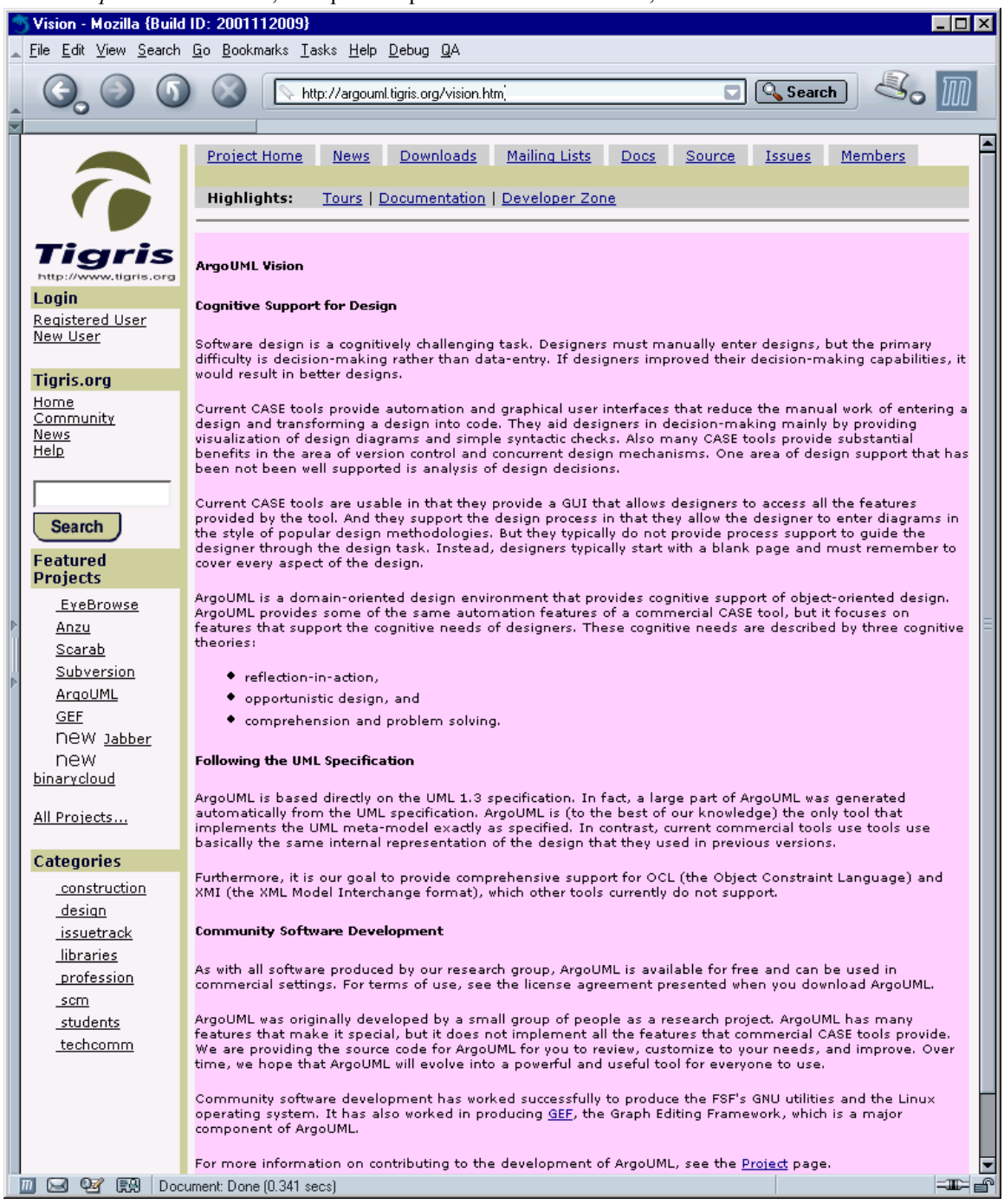


Exhibit 2. A software requirements vision statement highlighting community development as a software development objective (i.e., a non-functional requirement). Source: http://argouml.tigris.org/vision.html, July 2001.

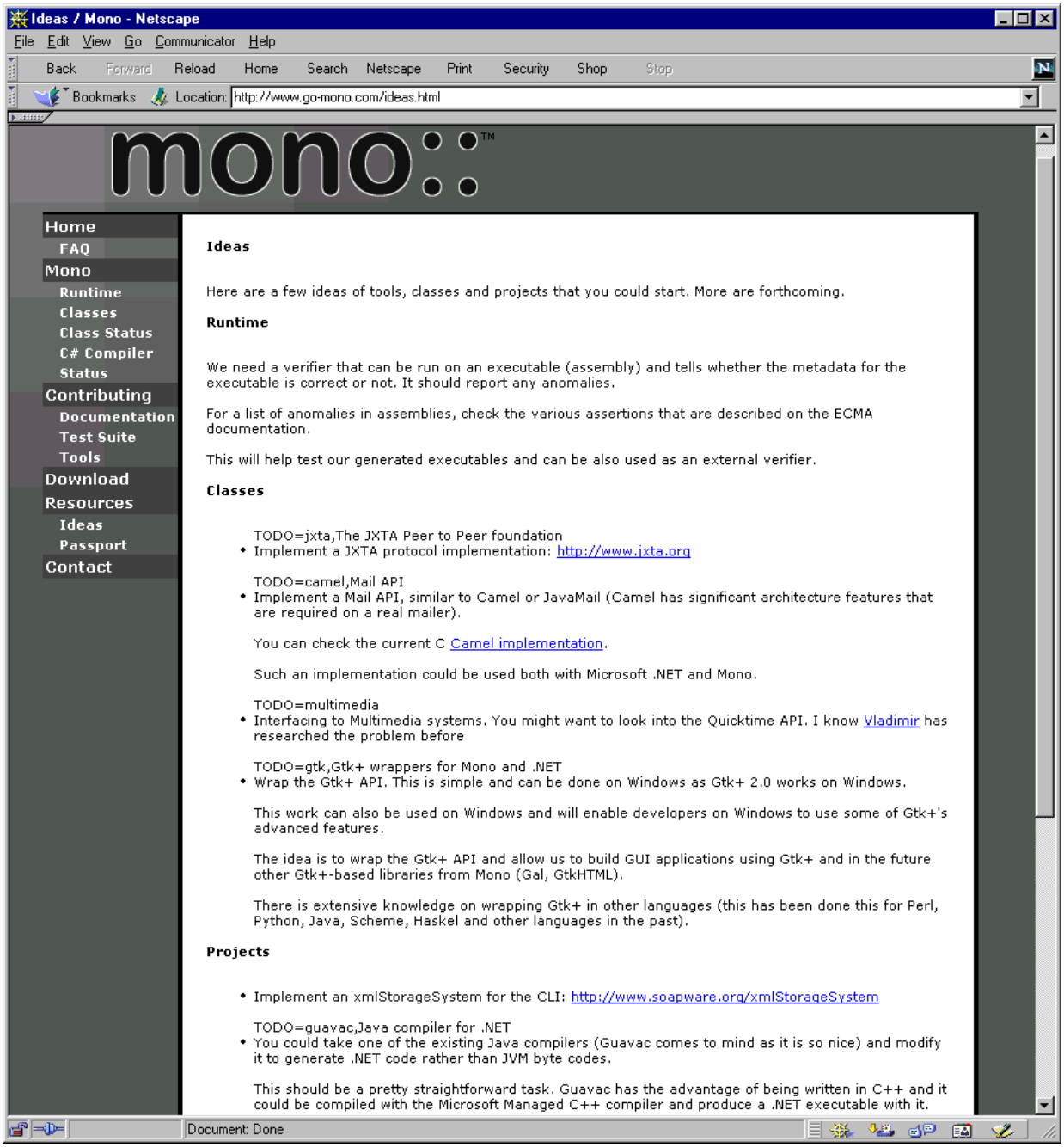


Exhibit 3: A non-functional requirement identifying a need for volunteers to become owners for software components (or classes) not yet bound to a developer. Source: http://www.go-mono.com/ideas.html, July 2001. The contents on this page have since been updated to reflect remaining tasks needing contributors, as well as adding new tasks for development.

Exhibit 4. An asserted capability (in the center) that invites would-be open software game developers to make extensions of whatever kind they require among the various types of available extensions ("…create your own levels, mods, skins, models, and more"). Source: http://www.unrealtournament.com/editing

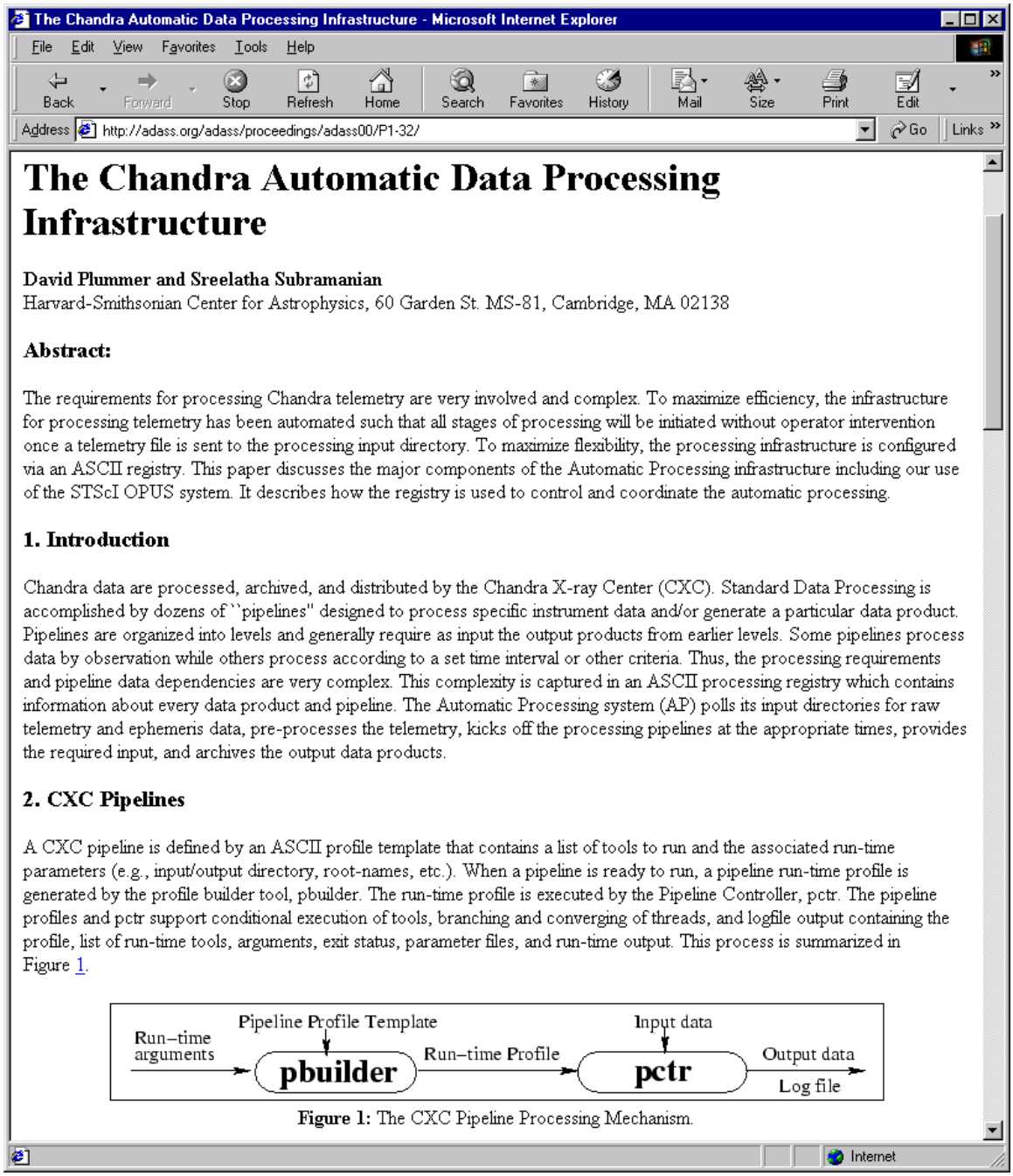


Exhibit 5. An asserted capability indicating that the requirements are very involved and complex, and thus require an automated, registry-based system software architecture for configuring dozens of application software pipelines. Source: [31].

The Chandra Automatic Data Processing Infrastructure - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites History Mail Size Print

Address http://adass.org/adass/proceedings/adass00/P1-32/ Go Links

4. Standard Pipeline Processing Threads

Figure 2 represents the series of pipelines that are run to process the Chandra data. Each circle represents a different pipeline (or related set of pipelines). Level 0 processing (De-commutation) will produce several data products that correspond to the different spacecraft components. Data from the various components of the spacecraft will follow different threads through the system. The arrows represent the flow of data as the output products of one pipeline are used as inputs to a pipe (or pipes) in the next level. Some pipelines are run on arbitrary time boundaries (as data are available) and others must be run on time boundaries based on observation interval start and stop times (which are determined in the level 0.5 pipe, OBI_DET).

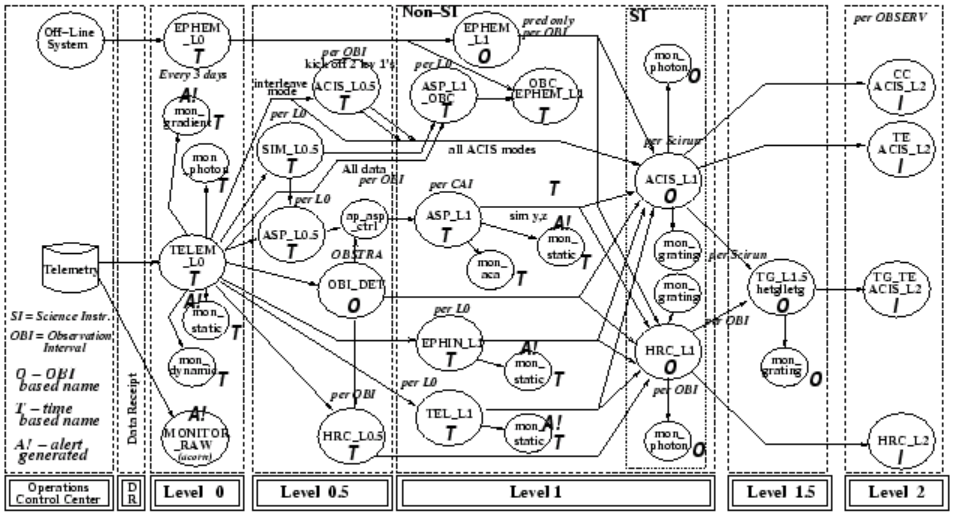


Figure 2: Standard Processing Threads.

5. Pipeline Processing Registry

The complete pipeline processing requirements for Chandra are very complex with many inter-dependencies (as can be seen in Figure 2). In order to run the pipelines efficiently in a flexible and automated fashion we configure the Automatic Processing system with a pipeline processing registry. We first register all the Chandra input and output data products. We can then capture the processing requirements and inter-dependencies by registering all the pipelines. Data products are registered with a File_ID, file name convention (using regular expressions), method for extracting start/stop times, and archive ingest keywords (detector, level, etc.). Pipelines are registered with a Pipe_ID, pipeline profile name, pbuilder arguments, kickoff criteria (detector in focal plane, gratings in/out, etc.), input and output data products (by File_ID), and method for generating the ``root'' part of output file names.

Internet

Exhibit 6. A specification of data-flow relationships among a network of software module pipelines that constitute the processing threads that must be configured in order to transform remotely sensed telemetry data into digital images of deep space objects. Source: [31]

CXCDS architecture is configured, in order to accept what is presented as plausible, accurate, and correct.

The notion that requirements for open software system are, in practice, analyzed via the reading of technical accounts as narratives, together with making sense of how such readings are reconciled with one's prior knowledge, is not unique to the X-ray astronomy software community. These same activities can and do occur in the other three communities. If one reviews the functional and non-functional requirements appearing in Exhibits 1-4, it is possible to observe that none of the descriptions appearing in these exhibits is self-contained. Instead, each requires the reader (e.g., a developer within the community) to closely or casually read what is described, make sense of it, consult other materials or one's expertise, and trust that the description's author(s) are reliable and accountable in some manner for the open software requirements that have been described [15, 29]. Analyzing open software requirements entails little if any automated analysis, formal reasoning, or visual animation of software requirements specifications [cf. 28]. Yet, participants in these communities are able to understand what the functional and non-functional requirements are in ways that are sufficient to lead to the ongoing development and routine use of various kinds of open software systems.

5.3 Requirements specification and modeling vs. continually emerging webs of software discourse

If the requirements for open software systems are asserted rather than elicited, how are these requirements specified or modeled? In examining data from the four communities, it is becoming increasingly apparent that open software requirements can emerge from the experiences of community participants through their email and bboard discussion forums (see Exhibit 1 for example). These communication messages in turn give rise to the development of narrative descriptions that more succinctly specify and condense into a web of discourse about the functional and non-functional requirements of an open software system. This discourse is rendered in descriptions that can be found in email and discussion forum archives, on Web pages that populate community Web sites, and in other informal software descriptions that are posted, hyperlinked, or passively referenced through the assumed common knowledge that community participants expect their cohorts to possess.

In Exhibit 5 from the X-ray and deep space imaging software community, we see passing reference in the opening paragraph to "the requirements for processing Chandra telemetry (imaging data) are very involved and complex." To comprehend and recognize what these involved and complex requirements are, community members who develop open software for such applications will often be astrophysicists (generally with Ph.D. degrees), and rarely would be simply a competent software engineering professional. Subsequently, the astrophysicists that develop software in this community do not need to recapitulate any software system requirement that would be due to the problem domain (astrophysics). Instead, community members are already assumed to have mastery over such topics prior to software development, rather than encountering problems in their understanding of astrophysicists arising from technical problems in developing, operation, or functional enhancement of remote sensing or digital imaging software. Thus, for openness to be effective, a software developer in this community must be competent in the discourse of astrophysics, as well as with that for the tools and techniques used in developing open software systems.

Subsequently, spanning the four communities and the six exhibits, we begin to observe that the requirements for open software are specified in webs of discourse that reference or link:

- email or bboard discussion threads,
- system vision statements,
- ideas about system functionality and the non-functional need for volunteer developers to implement the functionality,
- promotional encouragement to specify and develop whatever functionality you need, which might also help you get a new job, and
- scholarly scientific research publications that underscore how the requirements of domain-specific software (e.g., for astronomical imaging), though complex, are understood without elaboration, since they rely on prior scientific/domain knowledge and tradition of open scientific research.

Each of these modes of discourse, as well as their Web-based specification and dissemination, is a continually emerging source of open software requirements from new contributions, new contributors or participants, new ideas, new career opportunities, and new research publications [cf. 37].

5.4 Requirements validation vs. condensing discourse that hardens and concentrates system functionality and community development

Software requirements are validated with respect to the software's implementation. Since open software requirements are generally not recorded in a formal SRS document, nor are these requirements typically cast in a mathematical logic, algebraic, or state transition-based notational scheme, then how are the software implementations to be validated against their requirements?

In each of the four communities, it appears that the requirements for open software are co-mingled with design, implementation, and testing descriptions and software artifacts, as well as with user manuals and usage artifacts (e.g., input data, program invocation scripts). Similarly, the requirements are spread across different kinds of electronic documents including Web pages, sites, hypertext links, source code directories, threaded email transcripts, and more. In each community, requirements are described, asserted, or implied informally. Yet it is possible to observe in threaded email/bboard discussions that community participants are able to comprehend and condense wide-ranging software requirements into succinct descriptions using lean media [39] that pushes the context for their creation into the background. Goguen [15] suggests the metaphor of "concentrating and hardening of requirements" as a way to characterize how software requirements evolve into forms that are perceived as suitable for validation. His characterization seems to quite closely match what can be observed in the development of requirements for open software. Subsequently, we find that requirements validation is an implicit by-product, rather than an explicit goal, of how open software requirements are constituted, described, discussed, cross-referenced, and hyperlinked to other informal descriptions of system and its implementations.

5.5 Communicating requirements vs. global access to open software webs

One distinguishing feature of open software associated with each of the four communities is that their requirements, informal as they are, are organized and typically stored in a persistent form that is globally accessible. This is true of community Web sites, site contents and hyperlinkage, source code directories, threaded email and bboard discussion forums, descriptions of known bugs and desired system enhancements, records of multiple system versions, and more. Persistence, hypertext-style organization and linkage, and global access to open software descriptions appear as conditions that do not receive much attention within the classic requirements engineering approaches, with few exceptions [9]. Yet, each of these conditions helps in the communication of open software requirements. These conditions also contribute to the ability of community participants or outsiders looking in to trace the development and evolution of software requirements both within the software development descriptions, as well as across community participants. This enables observers or developers to navigationally trace, for example, a web of different issues, positions, arguments, policy statements, and design rationales that support (e.g., see Exhibit 1) or challenge the viability of emerging software requirements [cf.5, 23]. Nonetheless, these traces appear to lack a persistent representation beyond the awkward "history" file of a Web browser.

Each of the four communities also communicates community-oriented requirements. These non-functional requirements may seem similar to those for enterprise modeling [28]. However, there are some differences, though they may be minor. First, each community is interested in sustaining and growing the community as a development enterprise [cf. 26]. Second, each community is interested in sustaining and growing the community's open software artifacts, descriptions, and representations. Third, each community is interested in updating and evolving the community's information sharing Web sites. In recognition of these community requirements, it is not surprising to observe the emergence of commercial efforts (e.g., SourceForge and CollabNet) that offer community support systems that are intended to address these requirements, such as is used in the ArgoUML community site, http://www.tigris.org, in the academic software design community.

5.6 Identifying a common foundation for the development of open software requirements

Based on the data and analysis presented above, it is possible to begin to identify what items, practices, or capabilities may better characterize how the requirements for open software are developed. This centers of the emergent creation, usage, and evolution of informal software descriptions as the vehicle for developing open software requirements. This is explored in the following section.

6. Informalisms for Open Software System Requirements

The functional and non-functional requirements for open software systems are elicited, analyzed, specified, validated, and managed through a variety of Web-based descriptions. These descriptions can be treated collectively as *software informalisms.* The choice to designate these descriptions as informalisms[10] is to draw a distinction between how the requirements of open software systems are described, in contrast to the recommended use of formal, logic-based requirements notations ("formalisms") that are advocated in traditional approaches [10, 17, 22, 28]. In the four communities examined in this study, software informalisms appear to be the preferred scheme for describing or representing open software requirements. There is no explicit objective or effort to treat these informalisms as "informal software requirements" that should be refined into formal requirements [9, 17, 22] within any of these communities. Accordingly, we can present an initial classification scheme that inventories the available types of software requirements informalisms that have been found in one or more of the four communities in this study. Along the way, we seek to identify some of the relations that link them together into more comprehensive stories, storylines, or intersecting story fragments that help convey as well as embody the requirements of an open source software system.

Eight types of software informalisms can be identified, and each has sub-types that can be identified as follows.

6.1 Community communications

The requirements for open software are asserted, read, discussed, condensed, and made accountable through a small set of computer-based communication tools and modalities. In the absence of co-located workplaces, a community's communication infrastructure serves as the "place" where software requirements engineering work is performed, and where requirements artifacts are articulated, refined, stored, or discarded. These communication systems, appear in the form of: (a) messages placed in a Web-based bboard discussion forums; (b) email list servers; (c) network news groups; or less frequently in (d) Internet-based chat (instant messaging)[11]. Messages written and read through these systems, together with references or links to other messages or software webs, then provide some sense of context for how to understand messages, or where and how to act on them.

6.2 Scenarios of usage as linked Web pages

Open software developers who do not meet face-to-face create, employ, read, and revise shared mental constructions of how a given system is suppose to function. Since shared understanding must occur at a distance in space or time, then community participants create artifacts like screenshots, guided tours, or navigational click-through sequences (e.g., "back", "next" Web page links) with supplementary narrative descriptions in attempting to convey their intent or understanding of how the system operates, or how it appears to a user when used. This seems to occur when participants find it simpler or easier to explain what is suppose to happen or be observable at the user interface with pictures (or related hypermedia) than with just words. Similarly, participants may publish operational program execution scripts or recipes for how to develop or extend designated types of open software artifacts. These hypermedia scenarios of usage may serve a similar purpose to formally elicited and modeled Use Cases, though there is no apparent effort to codify these usage scenarios in such manner or notational form in any of the communities in this study.

---

[10] As Goguen [15] observes, formalisms are not limited to those based on a mathematical logic or state transition semantics, but can include descriptive schemes that are formed from structured or semi-structured narratives, such as those employed in Software Requirements Specifications documents.

[11] Instant messaging can be more widely observed in the networked computer game community in contrast to the academic software design and X-ray astrophysics community, where we are yet to find traces of instant messaging activities in support of open software development.

6.3 HowTo Guides

Online documents that capture and condense "how to" perform some behavior, operation, or function with a system, serve as a semi-structured narrative that assert or imply end-user requirements. "Formal" HowTo's descriptions include explicit declarations of their purpose as a HowTo and may be identified as a system tutorial. Community participants may seek these formal HowTo's when they need to add a system module or class structure, or contribute other resources or efforts to the open software project. In contrast, informal HowTo's may appear as a selection, composition, or recomposition of any of the proceeding. These informal HowTo guides may be labeled as a "FAQ"; that is, as a list of frequently asked questions about how a system operates, how to use it, where to find it's development status, who developed what, known bugs and workarounds, etc. However, most FAQs do not indicate how frequently any of the questions may have been asked, or if effort has been made to measure or track FAQ usage/reference.

6.4 External Publications

In each of the four communities in this study, there are external publications that describe open software available for consumption by the public or by community members. Most common among these are technical articles, while books are less common, though of growing popularity in the Internet/Web infrastructure and computer game community. Many developers find that books, especially those derived from composition and extension of other open software informalisms, are both a valuable and convenient source for recording, recontextualizing, and explaining the functional and non-functional requirements of an open software system, or how it was developed [e.g., 11, 14, 32].

On the other hand, professional articles that inform interested readers or promote the author's interests in a certain open software technology, help identify general functional and non-functional software requirements for these systems. These article may appear in trade publications, like the *Linux Journal* or *Game Developer* [7]. Academic articles that are refereed and appear in conference proceedings or scholarly journals [25, 33, 35], serve a similar purpose as professional articles, though usually with more technical depth, theoretical recapitulation, analytical detail, and extensive bibliography of related efforts. However, it may be the case that readers of academic research papers bring to their reading a substantial amount of prior domain knowledge. This expertise may enable them to determine what open software requirements being referenced may be obvious from received wisdom, versus those requirements that are new, innovative, or otherwise noteworthy.

6.5 Open Software Web Sites and Source Webs

As already suggested, open software is most easily found on the Web or Internet. Neither information infrastructure is an absolute necessity for open software. However, such global infrastructure is an enabler of open software communities, processes, and practices. But open software communities take advantage of a community Web site as an information infrastructure for publishing and sharing open descriptions of software in the form of Web pages, Web links, and software artifact content indexes or directories. These pages, hypertext links, and directories are community information structures that serve as a kind of organizational memory and community information system. Such a memory and information system records, stores, and retrieves how open software systems and artifacts are being articulated, negotiated, employed, refined, and coordinated within a community of collaborating developer-users [5, 23, 1].

Web pages in each of the four open software communities include *content* that incorporates text, tables or presentation frames, diagrams, or navigational images (image maps) to describe their associated open software systems. This content may describe vision statements, assert system features, or otherwise characterize through a narrative, the functional and non-functional capabilities of an open software system. Whether this content can be considered a software requirements specification document is unclear, since to do so would seem to allow almost any document with a narrative about some software system to be considered as an SRS document, and thus a formal requirements specification.

Web content that describes an open software system often comes with many embedded Web *links*. These links associate content across Web pages, sites, or applications. These links may simply denote traversal links (i.e., "goto" links to related software components, for example, go to the source code for a named procedure call) or other source/Web content where there is no further explicit meaning or semantics

assigned to the link. Alternatively, they may serve as enumerated navigational index links (e.g., site indexes) that helps direct a community participant (or outsider) to find their way around the community, its community information base, and the community's open source code base. Beyond this, they can serve as links to implied "helper applications" or tools invoked by navigational access to remotely served file types that are registered on the client, and associated with external application programs or plug-in programs that are invoked when selected or traversed [26, 27].

Each of the open software communities in this study provides access to Web-based source code directories, files, or compositions for download, build and/or installation. These directories and files contain operational software or open source code, as well as some related support files or Web pages. Source code denote requirements implementations, rather than requirements specifications. Program execution scripts, which take the form of C-shell, Tcl, Perl, Python, or Java scripts on Linux/Unix systems, may be employed for invoking other system modules. These execution scripts are functional in the sense that they invoke or cause system behavior that is implemented in the open software source code. Since scripts are generally platform specific, they effectively impose their own functional requirements on an implemented system. Thus, it is not surprising to find examples in each community for Web pages or files that describe these requirements explicitly, while the requirements of the open software system whose behavior in under control of the script has its requirements left implicit. The same kinds of concerns and explication of functional requirements is also found for make files (compilation or build scripts), CVS files that specify, control and synchronize concurrent software versions, and deployment-installation compositions (e.g., "tarballs" or zip files) for coordinating shared software production and distribution [14].

6.6 Software bug reports and issue tracking

One of the most obvious and frequent types of discourse that appears with open software systems is discussion about operational problems with the current version of the system implementation. Bugs and other issues (missing functionality, incorrect calculation, incorrect rendering of application domain constructs, etc.) are common to open software, much like they are with all other software. However, in an open software development situation, community participants rely on lean communication media like email, bug report bboards, and related issue tracking mechanisms to capture, rearticulate, and refine implicit, mis-stated, or unstated system requirements [39]. We find the capabilities of bug report or issue-tracking systems like Bugzilla in the Internet/Web infrastructure community, are also appearing in the academic software design community and networked game communities. In contrast, software developers in X-ray astrophysics community still rely on threaded email discussion lists to manage their (re) emerging requirements, often with relatively few message postings.

6.7 Traditional software system documentation

Open software systems are not without online system documentation or documentation intended to be printed in support of end-users or developers. For all of the systems examined in this study, it was possible to locate online man pages or help pages that describe commands and command parameters for how to invoke or use a system. Similarly, it was possible to locate online user manuals for most of these systems. It was apparent in both situations that online documentation was usually dated, and subsequently inconsistent with current functional capabilities or system commands.

This may just be what should be expected of both closed and open software systems. However, it is apparent that there are many other information resources, that is, the other software informalisms, that are available to developers and end-users to help them detect or resolve inconsistencies in such documentation.

The overall set of software informalisms serve as a context for reconstructing what a system's functional requirements were, are, or can be. Thus, while open software manuals are not necessarily any better or worse than for other software, the context for their use may enable the typical inconsistencies one encounters to be more readily resolved. Subsequently, there are few incentives to make online manuals or help files for open software systems any better than the minimum needed to assist an unfamiliar user or developer to get started, or where to look for further help. Good enough documentation is good enough.

6.8 Software extension mechanisms and architectures

The developers of software systems in each of the four communities seek to keep their systems open through provision of a variety of extension mechanisms and architectures. These are more than just open application program interfaces (APIs); generally they represent operational mechanisms or capabilities. The extensions include embedded scripting languages, such as UnrealScript for *Unreal Tournament*; and Perl/Python for Internet/Web infrastructure applications. Open software architectures accommodate operational plug-in modules, as in the case for the Apache web server and Chandra system infrastructure. Other open architectural schemes accommodate reconfigurable processing pipelines, like the Chandra Data Processing Infrastructure. Finally, in the networked computer game community, we see game vendors providing tools and utilities to assist advanced users so that they can develop their own extensions or custom programs in order to keep an open software system alive and continuously evolving.

Whether these mechanisms and architectures can or should be treated as software formalisms is perhaps subject to debate. However, across the four community, it is apparent that software extensions mechanisms and extensible software architectures contribute to, as well as enable, the continuing emergence open software requirements.

Overall, it appears that none of these software informalisms would defy an effort to formalize them in some mathematical logic or analytically rigorous notation. Nonetheless, in the four software communities examined in this study, there is no perceived requirement for such formalization, nor no unrecognized opportunity to somehow improve the quality, usability, or cost-effectiveness of the open software systems, that has been missed. If formalization of these software benefits has demonstrable benefit to members of these communities, beyond what they already realize from current practices, these benefits have yet to be articulated in the discourse that pervades each community.

7. Understanding open software requirements

In open software development projects, requirements engineering efforts are *implied activities* that routinely emerge as a by-product of community discourse about what their software should or should not do, as well as who will take responsibility for realizing such requirements. Open software system requirements appear in the form of situated discourse within private and public email discussion threads, emergent artifacts (e.g., source code fragments included within a message) and dialectical social actions that negotiate interest, commitment, and accountability [15, 37]. More simply, traditional requirements engineering activities do not have first-class status as an assigned or recognized task within open software development communities. Similarly, there are no software engineering tools used to support the capture, negotiation, and cost estimate (e.g., level of effort, expertise/skill, and timeliness) of open software development efforts, though each of these activities occurs regularly but informally.

Open software systems may be very reliable and high quality in their users' assessments. Nonetheless requirements do exist, though finding or recognizing them demands familiarity and immersion within the community and its discussions. This of course stands in contrast to efforts within the academic software engineering or requirements engineering community to develop and demonstrate tools for explicitly capturing requirements, negotiating trade-offs among system requirements and stakeholder interests, and constructive cost estimation or modeling [e.g., 3]. Furthermore, in open software systems, the developers are generally end-users of the systems they develop, whereas in traditional software requirements engineering efforts, developers and users are distinct, and developers tend not to routinely use the systems they develop. Perhaps this in why open software systems can suffice with reliance on software informalisms, while traditional software engineering efforts must struggle to convert informal requirements into more formal ones.

Developing open software requirements is a *community building process* that must be institutionalized both within a community and its software informalisms to flourish [30, 36]. In this regard, the development of requirements for open software is not a traditional requirements engineering process, at least, not yet. It is instead socio-technical process that entails the development of constructive social relationships, informally negotiated social agreements, and a commitment to participate through sustained contribution of software discourse and shared representations. Thus, community building and sustaining participation are essential

and recurring activities that enable open software requirements and system implementation to emerge and persist without central corporate authority.

Open software Web sites serve as hubs that centralize attention for what is happening with the development of the focal open software system, its status, participants and contributors, discourse on pending/future needs, etc. Furthermore, by there very nature, open software Web sites (those accessible outside of a corporate firewall) are generally global in reach and accessibility. This means the potential exists for contributors to come from multiple remote sites (geographic dispersion) at different times (24/7), from multiple nations, potentially representing the interests of multiple cultures or ethnicity. All of these conditions point to new kinds of requirements—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. These requirements may entail both functional and non-functional requirements, but they will most typically be expressed using open software informalisms, rather than using formal notations based on some system of mathematical logic.

8. Conclusions

The paper reports on a study that investigates, compares, and describes how the requirements engineering processes occurs in open source software development projects found in different communities. A number of conclusions can be drawn from the findings presented.

First, this study sought to discover and describe the practices and artifacts that characterize how the requirements for developing open software systems are developed. Perhaps the processes and artifacts that were described were obvious to the reader. This might be true for those scholars and students of software requirements engineering who have already participated in open software projects. However, advocates of open source software do not identify or report on the processes described here [11, 29, 32]. Thus, we must ask what is obvious to whom, and on what source of knowledge or experience is it based? For the majority of students who have not participated, it is disappointing to not find such descriptions, processes, or artifacts within the classic or contemporary literature on requirements engineering [10, 17, 22, 28]. In contrast, this study sought to develop a baseline characterization of the how the requirements process for open software occurs and the artifacts (and other mechanisms).

Given such a baseline of the "as-is" process for open software requirements engineering, it now becomes possible to juxtapose one or more "to-be" prescriptive models for the requirements engineering process, then begin to address what steps are needed to transform the as-is into the to-be [34]. Said differently, what is the process that gets open software development from "here to there", from the as-is to the to-be? Such a position provides a basis for further studies which could examine how to redesign open software practices into those closer to that advocated by classic or contemporary scholars of software requirements engineering. This would enable students or scholars of software requirements engineering, for example, to determine whether or not open source software development would benefit from more rigorous requirements elicitation, analysis, and management, and if so, how. Similarly, it might help determine when a software requirements process that relies on the use of informalisms will be more/less effective than one rooted in formalisms.

Second, this report describes a new set of processes that constitute how open software requirements are developed or engineered in the form of a narrative model (cf. Section 5). We can therefore begin a follow-on step to develop a more comprehensively detailed model of these processes [20, 38], which in turn can be further analyzed or simulated if codified as a computational process model [24, 27, 34]. Such a process model could then be aligned and combined with those for traditional requirements engineering (cf. Section 4). Such an integration of processes would broaden the scope of requirements engineering to include open software development, not as an example of poor quality software engineering, but as an alternative to the dominant tradition advocated by contemporary software requirements experts and scholars. Clearly, the development of open software systems entails social and technical relations that differ from those advocated within traditional software or requirements engineering texts. It is unclear what kinds of software systems are most amenable to an open source approach, and which still seem to require the struggle to formalize traditional software requirements specifications.

Third, this study reports on the centrality and importance of open software requirements processes and software informalisms to the development of open software systems, projects, and communities. This result might be construed as an advocacy of the 'informal' over the 'formal' in how software system requirements are or should be developed and validated, though it is not so intended. Instead, attention to software informalisms used in open software projects, without the need to coerce or transform them into more mathematically formal notations, raises the issue of what kinds of *engineering virtues* should be articulated to evaluate the quality, reliability, or feasibility of open software system requirements so expressed. For example, traditional software requirements engineering advocates the need to assess requirements in terms of virtues like consistency, completeness, traceability, and correctness [10, 17]. From the study presented here, it appears that open software requirements artifacts might be assessed in terms of virtues like encouragement of community building; freedom of expression and multiplicity of expression with software informalisms; readability and ease of navigation; and implicit versus explicit structures for organizing, storing and sharing open software requirements. "Low" measures of such virtues might potentially point to increased likelihood of a failure to develop a sustainable open software system. Subsequently, improving the quality of such virtues for open software requirements may benefit from tools that encourage community development; social interaction and communicative expression; software reading and comprehension; community hypertext portals and Web-based repositories. Nonetheless, resolving such issues is an appropriate subject for further study.

Overall, open software development practices are giving rise to a new view of how complex software systems can be constructed, deployed, and evolved. Software informalisms and their corresponding software applications/tools are not yet part of, nor integrated with, the traditional requirements engineer's toolset. Open software development does not adhere to the traditional engineering rationality or virtues found in the legacy of software engineering life cycle models or prescriptive standards.

The development open software system requirements is inherently and undeniably a complex web of socio-technical processes, development situations, and dynamically emerging development contexts [2, 15, 20, 37, 38]. In this way, the requirements for open software systems continually emerge through a web of community narratives. These extended narratives embody discourse that is manifest through an open software requirements engineering process. Participants in this process capture in persistent, globally accessible, open software informalisms that serve as their organizational memory [1], hypertextual issue-based information system [5, 23], and a networked community environment for information sharing, communication, and social interaction [18, 30, 36, 37]. Consequently, ethnographic methods are needed to elicit, analyze, validate, and communicate what these narratives are, what form they take, what practices and processes give them their form, and what research methods and principles are employed to examine them [15, 16, 19, 20, 28, 38]. Employing these methods reveals what is involved in this open software process, and how developer-users participate to create their discourse using software requirement informalisms in the course of developing the open software systems they seek to use and sustain. This report thus contributes a new study of this kind.

Acknowledgements

The research described in this report is supported by a grant from the National Science Foundation #IIS-0083075, and from the Defense Acquisition University by contract N487650-27803. No endorsement implied. Mark Ackerman at the University of Michigan, Mark Bergman, Xiaobin Li, and Margaret Elliott, at the UCI Institute for Software Research, and also Julia Watson at The Ohio State University are collaborators on the research project described in this paper.

8. References

1. ACKERMAN, M.S. and HALVERSON, C.A.: 'Reexamining Organizational Memory', *Communications ACM*, **43**, (1), pp. 59-64, January 2000.
2. ATKINSON, C.J.: 'Socio-Technical and Soft Approaches to Information Requirements Elicitation in the Post-Methodology Era', *Requirements Engineering*, **5**, pp. 67-73, 2000.
3. BOEHM, B., EGYED, A., KWAN, J. PORT, D., SHAH, A., AND MADACHY, R.: 'Using the WinWin Spiral Model: A Case Study', *Computer*, **31**, (7), pp. 33-44, 1998.

4. BOWKER, G.C. and STAR, S.L.: '*Sorting Things Out: Classification and Its Consequences'*, MIT Press, Cambridge, MA, 1999.
5. CONKLIN, J. and BEGEMAN, M.L.: 'gIBIS: A Hypertext Tool for Effective Policy Discussion', *ACM Transactions Office Information Systems*, **6**, (4), pp. 303-331, October 1988.
6. COOK, J.E., VOTTA, L.G., and WOLF, A.L.: 'Cost-effective analysis of in-place software processes', *IEEE Transactions Software Engineering*, **24**, (8), pp. 650-663, 1998.
7. CLEVELAND, C.: 'The Past, Present, and Future of PC Mod Development', *Game Developer*, pp. 46-49, February 2001.
8. CURTIS, B., KELLNER, M.I. and OVER, J.: 'Process modeling', *Communications ACM*, **35**, (9), pp. 75- 90, 1992.
9. CYBULSKI, J.L. and REED, K.: 'Computer-Assisted Analysis and Refinement of Informal Software Requirements Documents', *Proceedings Asia-Pacific Software Engineering Conference* (APSEC'98), Taipei, Taiwan, R.O.C., pp. 128-135, December 1998.
10. DAVIS, A.M.: '*Software Requirements: Analysis and Specification'*, Prentice-Hall, 1990.
11. DIBONA, C. OCKMAN, S. and STONE, M.: '*Open Sources: Voices from the Open Source Revolution'*, O'Reilly Press, Sebastopol, CA, 1999.
12. FIELDING,R.T.: 'Shared Leadership in the Apache Project', *Communications ACM,* **42**, (4), pp. 42-43, April 1999.
13. FLAKE, G.W., LAWRENCE, S., and GILES, C.L.: 'Efficient Identification of Web Communities', *Proc. Sixth Intern. Conf. Knowledge Discovery and Data Mining*, (ACM SIGKDD-2000), Boston, MA, pp. 150-160, August 2000.
14. FOGEL, K.: '*Open Source Development with CVS*'. Coriolis Press, 1999.
15. GOGUEN, J.A.: 'Formality and Informality in Requirements Engineering (Keynote Address)', *Proc. 4th. Intern. Conf. Requirements Engineering,* pp. 102-108, IEEE Computer Society, 1996.
16. HINE, C.: '*Virtual Ethnography'*, SAGE Publishers, London, 2000.
17. JACKSON, M.: '*Software Requirements & Specifications: Practice, Principles, and Prejudices'*, Addison-Wesley Pub. Co., Boston, MA, 1995.
18. KIM, A.J.: '*Community-Building on the Web: Secret Strategies for Successful Online Communities'*, Peachpit Press, 2000.
19. KLEIN, H. AND MYERS, M.D.: 'A Set of Principles for Conducting and Evaluating Intrepretive Field Studies in Information Systems', *MIS Quarterly*, **23**, (1), pp. 67-94, March 1999.
20. KLING, R. and SCACCHI, W.: 'The Web of Computing: Computer technology as social organization'. In M. Yovits (ed.), *Advances in Computers*, **21**, pp. 3-90. Academic Press, New York, 1982.
21. KOCH, S. and SCHNEIDER, G.: 'Results from software engineering research into open source development projects using public data', *Diskussionspapiere zum Taetigkeitsfeld Informationsverarbeitung und Informationswirtschaft*, H.R. Hansen und W.H. Janko (Hrsg.), Nr. 22, Wirtschaftsuniversitaet Wien, 2000.
22. KOTONYA, G. and SOMMERVILLE, I.: '*Requirements Engineering: Processes and Techniques',* John Wiley and Sons, Inc, New York, 1998.
23. LEE, J.: 'SIBYL: a tool for managing group design rationale', *Proceedings of the Conference on Computer-Supported Cooperative Work*, Los Angeles, CA, ACM Press, pp. 79-92, 1990.
24. MI, P. and SCACCHI, W.: ' * A Meta-Model for Formulating Knowledge-Based Models of Software Development', *Decision Support Systems*, **17**, (4), pp. 313-330, 1996.
25. MOCKUS, A., FIELDING, R.T., and HERBSLEB, J.: 'A Case Study of Open Software Development: The Apache Server'*, Proc. 22nd. International Conference on Software Engineering*, Limerick, IR, pp. 263-272, 2000.
26. NOLL, J. and SCACCHI, W.: 'Supporting Software Development in Virtual Enterprises'. *J. Digital Information*, **1**, (4), February 1999, http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll/
27. NOLL, J. and SCACCHI. W.: 'Specifying Process-Oriented Hypertext for Organizational Computing', *J. Network and Computer Applications*, **24**, (1), pp. 39-61, 2001.
28. NUSEIBEH, R. and EASTERBROOK, S.: 'Requirements Engineering: A Roadmap', in A. Finkelstein (ed.), *The Future of Software Engineering*, ACM and IEEE Computer Society Press, http://www.softwaresystems.org/future.html, 2000.
29. PAVLICEK, R.: '*Embracing Insanity: Open Source Software Development'*, SAMS Publishing, Indianapolis, IN, 2000.

30. PREECE, J.: '*Online Communities: Designing Usability, Supporting Sociability'*. Chichester, UK: John Wiley & Sons, 2000.
31. PLUMMER, D.A. and SUBRAMANIAN, S.: 'The Chandra Automatic Data Processing Infrastructure', in *ASP Conference. Series, **238**, Astronomical Data Analysis Software and Systems X*, in F. R. Harnden, Jr., F. A. Primini, & H. E. Payne (eds.), San Francisco: ASP, Paper #475, 2000.
32. RAYMOND, E.: '*The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary'*, O'Reilly and Associates, Sebastopol, CA, 2001.
33. ROBBINS, J. E. and REDMILES, D. F.: 'Cognitive support, UML adherence, and XMI interchange in Argo/UML', *Information and Software Technology*, **42**, (2), pp. 71-149, 25 January 2000.
34. SCACCHI, W.: 'Understanding Software Process Redesign using Modeling, Analysis and Simulation , *Software Process--Improvement and Practice*, **5**, (2/3), pp. 183-195, 2000.
35. SHORTRIDGE, K.: 'Astronomical Software--A Review', *ASP Conference. Series., **238**, Astronomical Data Analysis Software and Systems X,* in F. R. Harnden, Jr., F. A. Primini, & H. E. Payne (eds.), San Francisco: ASP, Paper #343, 2000.
36. SMITH, M. and KOLLOCK, P. (eds.): '*Communities in Cyberspace'*, Routledge, London, 1999.
37. TRUEX, D., BASKERVILLE, R. and KLEIN, H.: 'Growing Systems in an Emergent Organization', *Communications ACM*, **42**, (8), pp. 117-123, 1999.
38. VILLER, S. and SOMMERVILLE, I.: 'Ethnographically informed analysis for software engineers', *Int. J. Human-Computer Studies*, **53**, pp. 169-196, 2000.
39. YAMAGUCHI, Y., YOKOZAWA, M., SHINOHARA, T., and ISHIDA, T.: 'Collaboration with Lean Media: How Open-Source Software Succeeds', *Proceedings of the Conference on Computer Supported Cooperative Work,* (CSCW'00), pp. 329-338, Philadelphia, PA, ACM Press, December 2000.
40. ZELOKOWITZ, M.V. and WALLACE, D.: 'Experimental Models for Validating Technology', *Computer*, **31**, (5), pp. 23-31, May 1998.

Understanding Free/Open Source Software Development Processes

Walt Scacchi[1], Joseph Feller[2], Brian Fitzgerald[3], Scott Hissam[4], and Karim Lakhani[5]

[1]Institute for Software Research, University of California, Irvine, USA
[2]Business Information Systems, University College Cork, Ireland
[3]Computer Science and Information Systems, Limerick University, Ireland
[4]Software Engineering Institute, Carnegie-Mellon University, USA
[5]Sloan School of Management, Massachusetts Institute of Technology, USA
November 2005

Abstract

This article introduces a special issue of *Software Process—Improvement and Practice* focusing on processes found in free or open source software development (F/OSSD) projects. It seeks to provide a background review of research in this area through a review of selected empirical studies of F/OSSD processes. The results and findings from a survey of empirical studies of F/OSSD give rise to an interesting variety of opportunities and challenges for understanding these processes, which are identified along the way. Overall, what becomes clear is that studies of F/OSSD processes reveal a more diverse set of different types of processes than have typically been examined in conventional software development projects. The papers in this special issue further advance understanding of what processes characterize and shape F/OSSD.

Keywords: Open source software development, free software development, free/open source software processes

Introduction

This article explores patterns and processes that emerge in free/open source software development (F/OSSD) projects. F/OSSD is a relatively new way for building and deploying large software systems on a global basis, and differs in many interesting ways from the principles and practices traditionally advocated for software engineering [Feller, *et al*. 2005]. Hundreds of F/OSS systems are now in widespread use by thousands, or in some cases, millions of end-users. And some of these F/OSS systems (e.g., Mozilla Web browser, OpenOffice productivity suite, Eclipse and NetBeans interactive development environments, KDE and GNOME user interface packages, and most Linux distributions) entail millions of lines of source code. So what's going on here, and how do emerging F/OSSD processes build and sustain these different projects? How might new studies of these processes be used to explore what's new or different?

One of the more significant features of F/OSSD is the formation and enactment of complex software development processes performed by loosely coordinated software developers and contributors who may be globally dispersed. On some large F/OSSD projects, companies are assigning and paying software development staff to work on

F/OSSD projects as part of their job. In contrast, many other developers volunteer their time and skill to such effort, and may only work at their personal discretion rather than as assigned and scheduled. Further, these developers generally provide their own computing resources, and bring their own software development tools with them. However, most F/OSS 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. But how are successful F/OSSD projects and processes possible without regularly employed and scheduled software development staff, or without an explicit regime for software engineering project management? Why will software developers participate in F/OSSD projects? Why and how are large F/OSSD projects sustained? How are large F/OSSD projects coordinated, controlled or managed without a traditional project management team? What is it about the communications, roles, and artifacts that enable some projects to persist but not others? Why and how might these answers to these questions change over time? These are the kinds of questions raised in this article, and in the papers that follow in this Special Issue of *Software Process—Improvement and Practice*.

The remainder of this article is organized as follows. The next section provides further background on what F/OSSD is and what is already known about F/OSSD practices, based on both trade studies and systematic empirical studies. This survey focuses attention on identifying opportunities and challenges in understanding F/OSSD processes through empirical studies. Following this are brief overviews of each of the five papers that were selected for this Special Issue after a comprehensive submission, review, and selection effort. A final discussion then argues for why the software process research community may itself want to adopt F/OSSD practices for sharing and enabling others to modify and share explicit descriptions, representations, models, and related "software process source code" within and across the software, information systems, computer science, and social science research communities.

What is free/open source software development?

Free (as in freedom) software and open source software are often labeled or treated as the same thing. However, there are important differences between them with regards to the licenses assigned to the respective software, and to the beliefs/ideology of their practitioners for how and why software should be developed for sharing, modification, reuse, and redistribution. 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 2004], whereas OSSD is a software development methodology, according to free software advocates like Richard Stallman and the Free Software Foundation [Gay 2002]. However, free software is always available as OSS, but OSS is not always free software[1]. This is why it often is appropriate to refer to F/OSS or FLOSS (L for *Libre*, where the alternative term of "libre software"

[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.

has popularity in some parts of the world) in order to accommodate similar and often indistinguishable approaches to software development. Subsequently, for the purposes of this article, focus is directed at F/OSSD processes, rather than to software licenses and social movements associated with free or open source software, though each may impinge on F/OSSD processes.

F/OSSD is mostly not about software engineering, at least not as SE is portrayed in modern SE textbooks. Similarly, F/OSSD is not SE done poorly. Instead, F/OSSD is a different, somewhat orthogonal approach to the development of software systems where much of the development activity is openly visible, development artifacts are publicly available over the Web, and generally there is no formal project management regime, budget or schedule. F/OSSD is oriented towards the joint development of a community of developers and users concomitant with the software system of interest.

F/OSS developers have typically been end-users of the F/OSS they develop, although this appears to be changing somewhat. Similarly, many end-users often participate in and contribute to F/OSSD efforts by providing feedback, bug reports, and usability concerns. There is also widespread recognition that F/OSSD projects can produce high quality and sustainable software systems that can be used by thousands to millions of end-users [Mockus, Fielding, Herbsleb 2002].

Subsequently, it is reasonable to assume that F/OSSD processes are not necessarily of the same type, kind, or form found in SE projects that follow the processes described in modern SE textbooks. 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 F/OSSD methods. Subsequently, what is known about SE processes, a development organization's process capability, and how to improve it's development processes may not be equally applicable to F/OSSD processes, without some explicit rationale or empirical justification. Thus, it is appropriate to review some of what is known so far about F/OSSD.

Results from recent studies of F/OSSD

There are studies that offer some insight or findings on F/OSSD practices where each, in turn, reflects on different kinds of processes which are not will understood at this time. These are systematic empirical studies of F/OSSD projects using small/large research samples and analytical methods drawn from different academic disciplines. Both kinds of studies stand in contrast to the popular examination of F/OSSD practices offered by F/OSS advocates [e.g., DiBona 1999, Pavelicek 2000, Raymond 2001]. These popular treatments tend to be grounded in personal experiences of the authors, rather than through careful systematic study, though such experiences are valuable because they are often a source of insight or questions for further inquiry.

A number of Web-based repositories of research papers that report on studies on F/OSSD projects have begun to appear. Among them are those at MIT (opensource.mit.edu) with over 200 papers contributed, and at University College Cork in Ireland (opensource.ucc.ie) which features links or citations to multiple special issue journals

focusing on F/OSSD (e.g., *Information Systems Journal*, 11(4) and 12(1), 2001-2002, *IEE Proceedings—Software*, 149(1), 2002, *Research Policy*, 32(7), 2003, and *IEEE Software*, 21(1), 2004), and to proceedings from international workshops of OSS research (e.g., 1st through 5th Workshops on Open Source Software Engineering, held in conjunction with the International Conferences on Software Engineering, 2001-2005). Rather than attempt to survey the complete universe of studies in these collections, since the majority of these studies do not address software processes, the choice instead is to examine a set of studies[2] that raise interesting issues or challenging problems for software process research and practice.

One important qualifier to recognize is that the studies below examined carefully selected F/OSSD projects, or a sample of projects, so the results presented should not be assumed to apply to all F/OSSD projects, or to projects that have not been studied. Furthermore, it is important to recognize that F/OSSD is no silver bullet that resolves the software crisis. Instead it is fair to recognize that most of the nearly 100,000 F/OSSD projects associated with Web portals like SourceForce.org have very small teams of two or less developers [Madey 2004], and most projects are inactive or have yet to release any operational software. However, there are now at least a few thousand F/OSSD projects that are viable and ongoing, so that there is a sufficient universe of diverse F/OSSD projects to investigate, understand, as well as to model and simulate their software processes. 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 F/OSSD projects or processes.

Comparing F/OSSD and SE Processes

The first category of related studies seek to identify and compare software development processes found in F/OSSD projects with those described or prescribed for SE projects, rather than just the resulting software products [Paulson, *et al*. 2004]. Mockus, Fielding, and Herbsleb [2002], in one of the most cited studies of F/OSSD, briefly describe the processes accounting for the development of the Apache Web Server and the Mozilla Web Browser. However, such an account does not provide sufficient content to directly compare them to traditional SE processes. In contrast, Reis and Fortes [2002] provide one of the first in-depth examinations of the overall process accounting for the development of the Mozilla Web Browser. They identify different developer roles, tools being used, artifacts created, and activities performed, which potentially provides adequate information for modeling and comparing the process.

Scacchi [2002] provides a narrative description of the software requirements process found in a sample of F/OSSD projects and compares it to the requirements engineering process portrayed in modern SE textbooks. The F/OSSD projects examined span applications in application domains including Internet infrastructure, networked computer games, astrophysics, and academic software design research. In a related study [Scacchi 2004], he identifies differences in software processes for requirements and design,

[2] The papers included here address some software development process, such as the OSS design process, in the body of their paper (as found using search engines), or address topics that heretofore have not appeared in prior software process studies.

configuration management, evolution, project management, and software technology transfer from those in SE texts as found in comparative study of multiple F/OSSD projects of a common type, networked computer games. Further, in two additional studies, he examines data and models accounting for software evolution [cf. Lehman 2002] compared to those emerging for F/OSSD [Scacchi 2005a], and also emerging socio-technical processes found in F/OSSD projects that intermingle social (e.g., team, group, and individual) and technical development processes [Scacchi 2005b, Truex, *et al.* 1999]. All of these studies describe processes found in different F/OSSD projects using narrative descriptions or models. Furthermore, recent work describes how these processes have been modeled and simulated in a variety of notational, computational, and ethnographic schemes [Scacchi, *et al*., 2005c].

Finally, other recent effort to model and simulate F/OSSD processes of different kinds has begun to appear. Antoniades, *et al*. [2004] and Smith *et al*. [2004] both provide simulation models of processes accounting for the overall development or evolution of multiple F/OSSD projects. Both efforts rely on models expressed as continuous functions through either algebraic formulae or systems of equations. Such an approach to process simulation and modeling appears well matched to Systems Dynamics-based process simulation tools. In contrast, Jensen and Scacchi [2004, 2005] model and re-enact processes found in a small sample of OSSD projects using language-based process models and a process re-enactment simulator fololwing techniques developed for analyzing traditional SE processes [Noll 2001, Scacchi 1999]. The ability to re-enact F/OSSD processes provides an approach for how to independently examine and validate whether the captured process reflects the observed practice, as well as makes the modeled processes available for reuse, tailoring, improvement, or practice in other F/OSSD projects.

Motivating, joining, participating, and contributing to F/OSSD projects

One of the most common questions about F/OSSD projects is why software developers will join and participate in such efforts, often without pay for sustained periods of time. Accordingly, we might then ask whether or how the participation of developers affects what gets done in a F/OSSD project in terms of which processes are engaged, as well as understanding how processes for recruiting and integrating new developers into F/OSSD projects operate. A number of surveys of F/OSS developers [Ghosh 2000, Lakhani, *et al.* 2002, Hars 2002, Hann 2002, Hertel 2003] have begun to pose such questions, and the findings reveal the following.

First, F/OSS developers generally find the greatest benefit from participation is the opportunity *to learn and to share what they know* about software system functionality, design, methods, tools, and practices associated with specific projects or community leaders [Lakhani, *et al.*, 2002]. F/OSSD 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 [Nakakoji 2002, Huntley 2003, Ye 2003]. However, though much of the development work in F/OSSD projects is unpaid or volunteer, individual F/OSS developers often benefit with higher average wages and better employment opportunities (at present), compared to their peers lacking F/OSSD

experience or skill [Hann 2002, Lerner 2002].

Second, F/OSS developers appear to really enjoy their F/OSSD work [Hertel 2003], and to be recognized as trustworthy and reputable contributors [Stewart 2001]. F/OSS developers also self-select the technical roles they will take on as part of their participation in a project [Ye 2003, Gacek 2004], rather than be assigned to a role in a traditionally managed SE project, where the assigned role may not be to their liking.

Third, many F/OSS developers participate in and contribute to multiple F/OSSD projects. In one study, 5% of developers surveyed reported participating in 10 or more F/OSSD projects [Hars 2002]. However, a small group of core developers who control the architecture and direction of development typically develop the vast majority of F/OSS released by a project. Subsequently, most participants typically contribute to just a single module, though a small minority of modules may be include patches or modifications contributed by hundreds of contributors [Ghosh 2000].

Consequently, how and why software developers will join, participate in, and contribute to an F/OSSD project seems to represent a new kind of process affecting how F/OSS is developed and maintained [von Krogh 2003, Scacchi 2005b]. Subsequently, modeling and simulating what this process is, how it operates, and how it affects software development is an open research challenge.

Alliance formation, social networking, community development and software development

How does the gathering of individual F/OSS developers give rise to a more persistent project team or self-sustaining community? Through choices that developers make for their participation and contribution to an F/OSSD 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 F/OSSD Web sites and online discourse (e.g., threaded email discussions) [Monge 1998], and they find they share many technical competencies, values, and beliefs in common [Crowston 2002, Espinosa 2002, Elliott 2004]. This manifests itself in the emergence of an occupational network of F/OSS developers [Elliott 2003].

Becoming a central node in a social network of software developers that interconnects multiple F/OSS projects is also a way to accumulate social capital and recognition from peers. However, it also enables the merger of independent F/OSS 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 2004, Scacchi 2005b]. “Linchpin developers” [Madey 2004] participate in or span multiple F/OSSD projects. In so doing, they create alliances between otherwise independent F/OSSD projects [cf. Hars 2002]. Figure 1 depicts an example of a social network of 24 F/OSS developers within 5 F/OSS projects that are interconnected through two linchpin developers [Madey 2004]. Such interconnection enables small F/OSS projects to come together as a larger social network with the critical mass [Marwell 1993] needed for their independent systems to be merged and collectively experience more growth in size, functionality, and user base.

F/OSSD Web sites also serve as hubs that centralize attention for what is happening with the development of the focal F/OSS system, its status, participants and contributors, discourse on pending/future needs, etc

Thus interesting problems arise when investigating how best to capture or represent the processes of alliance formation and inter-project social networking, and how such processes can be shown to facilitate or constrain F/OSSD activities, tool usage, and preference for which development artifacts are most valued by project participants.

Developing F/OSS systems is a community and project team building process that must be institutionalized within a community [Sharma 2002, Smith 1999, Preece 2000, Ye 2004] for its software informalisms (artifacts) and tools to flourish. Downloading, installing, and using F/OSS systems acquired from other F/OSS Web sites is also part of a community building process [Kim 2000]. Adoption and use of F/OSSD project Web sites are a community wide practice for how to publicize and share F/OSS project assets. These Web sites can be built using F/OSSD Web site content management systems (e.g., PhP-Nuke) to host project contents that can be served using F/OSS Web servers (Apache), database systems (MySQL) or application servers (JBoss), and increasingly accessed via F/OSS Web browsers (Mozilla and Firefox). Furthermore, ongoing F/OSSD projects may employ dozens of F/OSS 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 F/OSS application in development. These projects similarly employ asynchronous systems for project communications that are persistent, searchable, traceable, public and globally accessible [Yamauchi 2000].

F/OSS systems, hyperlinked artifacts and tools, and project Web sites serve as venues for socializing, building relationships and trust, sharing and learning with others. Community building, alliance forming, and participatory contributing are essential and recurring activities that enable F/OSSD projects to persist without central corporate authority. Linking people, systems, and projects together through shared artifacts and sustained online discourse enables a sustained socio-technical community, information infrastructure [Jensen 2005], and network of alliances [Monge 1998] to emerge.

For this reason, problems arise when investigating how best to capture and represent the F/OSSD processes that facilitate and constrain the co-development and co-evolution of F/OSS 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, they must be captured, understood, modeled, and simulated/re-enacted as integrating and intertwining processes.

Software evolution in a multi-project software ecosystem

As noted above, many F/OSSD 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 [Highsmith 2002], whereby ongoing development and evolution of one F/OSS system gives rise to propagated effects, changes, or vulnerabilities in one or more of the projects linked to it [Jensen 2005]. These interdependencies are most apparent when F/OSSD project share source code modules or components. In such situations, the volume of source code of an individual F/OSSD project may appear to grow at a super-linear or exponential rate [Scacchi 2005a, Schach 2002, Smith, *et al*., 2004]. Such an outcome, which economists and political scientists refer to as a "network externality" [Ostrom 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 might challenge the well-established laws of software evolution [Lehman 1980, 2002]. Thus, software evolution in a multi-project F/OSS ecosystem is a process of co-evolution of interrelated and interdependent F/OSSD projects, people, artifacts, tools, code, and project-specific processes.

Software evolution in a multi-project F/OSS ecosystem also suggests attending to social or technological mechanisms that provide some form of "natural selection". In biological ecosystems, natural selection provides an account for why some species flourish and adapt in response to environmental pressures, such as shortage of food sources or the rise of new predators, while other species that don't adapt progressively disappear or become extinct. In F/OSS ecosystems, a diversity of software system variants often appear as distinct projects. For example, there are a number of F/OSS operating systems with projects based on variants of the Unix operating system—Linux, FreeBSD, OpenBSD, Darwin, GNU Hurd, etc., and each of these may have multiple sub-variants (or forked distributions) like Debian GNU/Linux, SUSE Linux, Red Hat Linux, and hundreds of others. Web browsers, software build/make tools, database management systems, file management utilities, content management systems, and others types of software systems can be found by the dozens, perhaps reflecting their development in different F/OSS ecosystem niches. Similarly, one can readily find at F/OSS project portals like SourceForge.net, Freshmeat.net or Savannah.gnu.org, multiple projects developing the same type of software system, but with variations in software architecture, choice of functional components, choice of programming language, and project contributors. However, in some software domains, a dominant software system and project has emerged to effectively displace alternative variants by a large majority, like the Apache Web server, though such dominance has not completely eliminated the contending alternative F/OSS project efforts. As such, accounting for such evolutionary adaptation in response to emerging technological opportunities (new tools) or limited access to more established F/OSS projects or core developers is thus a challenge for those seeking to understand the processes of software evolution across a software ecosystem [cf. Lehman 2002, Nakakoji 2002, Smith, *et al.*, 2004, Ye 2004].

Last, it seems reasonable to observe that the world of F/OSSD is not the only place where multi-project software ecosystems emerge, as software sharing or reuse within traditional software development enterprises is common [cf. Highsmith 2002, Jensen 2005]. However, the process of the co-evolution of software ecosystems found in either traditional or F/OSSD projects in mostly unknown. Thus, software co-evolution within an

F/OSS ecosystem represents an opportunity for research that investigates understanding such a software evolution process through studies supported by techniques for modeling and simulating co-evolving processes.

Overall, the sample of F/OSSD research studies and findings presented above reveals a number of interesting challenges for research in understanding F/OSSD processes. However, these studies are all grounded in an empirical basis where different types of processes are being examined in different types of F/OSSD projects of varying sample size and data collection methodology. So the fundamental problem at hand is how to organize, reframe, and make clear what the challenges are in researching, improving, and practicing F/OSSD processes. The papers in this special issue help provide new insights and findings for better understanding the problem and challenges.

Papers Selected for the Special Issue

Five papers were selected as a result of the submission and review process from a pool of 21 submitted papers for inclusion in this Special Issue. In our first article, “Evaluation of Free/Open Source Software Products through Project Analysis,” David Cruz, Thomas Wieland, and Alexander Ziegler introduce a systematic approach for supporting a decision to incorporate F/OSS products into a larger context, such as a software or enterprise-wide system. The process of evaluating and integrating commercially available off-the-shelf software (often referred to as COTS software) has been written and described at length in academic and industrial literature. Often such evaluations are conducted to avoid unnecessary risks, including the technical [Hissam and Plakosh 1999] and the mission needs of the system to which the evaluated software is to be incorporated [Carney, *et al*., 2003]. Often, many of the same considerations apply to F/OSS products. However, as this article points out, there are additional and relevant aspects of an F/OSS project that produces the F/OSS product that should be taken into consideration. The authors nicely characterize these considerations into functional, technical, organizational, legal, economical and political aspects and thereby provide a broader perspective on F/OSS products when performing such evaluations.

In "Information Systems Success in Free and Open Source Software Development: Theory and Measures", Kevin Crowston, Hala Annabi, James Howison, and Chengetai Masango address a key gap in FLOSS research by seeking to define what "success" means in a FLOSS context. They derive a range of potential measures from the Information Systems (IS) literature, including system and information quality, user satisfaction, use experience, individual and organizational impacts, and then add a number of additional measures specific to the dynamics of the FLOSS development process. The list of measures was then refined, operationalized, and validated through empirical studies based in the SlashDot and SourceForge communities. The paper makes a welcome contribution by providing a solid instrument for the future evaluation of FLOSS processes, and one which will mature and increase in its utility with further use.

David Nichols and Michael Twidale report on their study of “Usability Processes in Open Source Software.” They describe mechanisms, techniques, and technology used in F/OSS projects like Mozilla and GNOME. Both of these projects develop systems that include

substantial user interface components and functionality, and so they are primary candidates to consider how F/OSS development processes and practices embrace or ignore usability concerns. They use examples drawn from bug reporting and discussion systems to highlight both the current practice, and how it might be revised to realize higher quality F/OSS development outcomes and easier to use application systems. This in turn may give rise for the recognizing how F/OSS projects needs to both address their ease of development (or "developability") and the usability of the resulting software systems. This is particularly true, as Nichols and Twidale observe, when developers and users are geographically dispersed, have limited resources to affect current processes, and may lack easily accessible sources of expertise about the functionality of the systems being developed, as well as how to make such functionality easy to use.

Douglas Schmidt and colleagues at Vanderbilt University and University of Maryland at College Park investigate "Techniques and Processes for Improving the Quality and Performance of Open Source Software." Their research complements the work of Crowston, *et al*., in this issue, but whereas that paper sought to define success, Schmidt, *et al.*, tackle the more specific question of software quality. They describe some of the challenges associated with FLOSS, and explore the ways in which quality assurance (QA) processes that are specifically designed for FLOSS processes can help address these challenges. They support their work with empirical examinations of FLOSS projects using these QA processes, and conclude with extremely practical findings of direct benefit to FLOSS practitioners.

Katherine Stewart and Sanjay Gosain examine "The Moderating Role of Development Stage in Affecting Free/Open Source Software Project Performance." This is an important contribution towards understanding how social factors like team trust and ideology interact with the development process of a project and impact objective and subjective outcomes like task completion, number of developers mobilized and perceived effectiveness. Using data from 67 F/OSS communities they show that the dynamics of performance change as a project moves through various development stages. Their results suggest that objective measures of project performance tend to improve over time and with increases in development stage, while subjective assessments depend to a greater extent on the project administrators' experience. For example, the importance of trust in teams varies based on the development stage of the project and the performance criteria. Overall they have presented a sophisticated view of the interaction between the social and technical factors in a F/OSSD project throughout its development process.

Discussion

F/OSSD projects represent and offer new publicly available data sources of a size, diversity, and complexity not previously available for research, on a global basis. Software process research and application has traditionally relied on an empirical basis in real-world processes for analysis, validation, or improvement. 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. In contrast, F/OSSD projects and project repositories contain process data and product artifacts that can be collected, analyzed, shared, and re-analyzed in a free and open source

manner. The papers in this Special Issue draw upon publicly available data and artifacts in their analyses. F/OSSD therefore 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. F/OSSD is thus poised to alter the calculus of empirical software engineering, information systems, and perhaps even computer science, while software process research is an arena that can take advantage of such a historically new opportunity.

Finally, one important dimension that has not yet been addressed in this article is whether and how the software process research community might adopt F/OSSD practices themselves. For example, one traditional barrier to engaging students in software process studies is the paucity of free or low-cost modeling and simulation tools. Sharing one's software process models and simulations with colleagues is difficult at present, if they must buy new and unfamiliar tools. The ability to reuse, re-analyze, or extend a colleague's models or simulations is similarly limited. The community needs and should directly benefit from F/OSS process models, tools, and process data/model repositories that can be easily acquired or shared, studied, modified, and redistributed to the mutual benefit of all. Similarly, it can also be noted that it further serves the collective interest of the community to consider how to develop a globally shared and interoperable information infrastructure for data sharing, modeling, and simulating software processes[3]. This is true whether these processes are found in SE or F/OSSD projects. As a consequence, these are all opportunities for the software process research community to realize and pursue. After all, we are the ones who will benefit from efforts to develop such free (as in freedom) and open source resources, as well as further our collective learning and community building efforts.

Conclusions

Free and open source software development is emerging as an alternative approach for how to develop large software systems. New types and new kinds of software processes are emerging within F/OSSD projects, as well as new characteristics for development project success, when compared to those found in traditional industrial software projects and those portrayed in software engineering textbooks. As a result, F/OSSD offer new types and new kinds of processes to research, understand, improve and practice. Similarly, understanding how F/OSSD processes are similar to or different from SE processes is an area ripe for further research and comparative study. Many new research opportunities exist in the empirical examination, modeling, simulation, improvement, and practice of F/OSSD processes.

Through a survey of empirical studies of F/OSSD projects and other analyses presented in this article, it should be clear there are an exciting variety and diversity of opportunities for new research aimed at understanding and improving the practice of F/OSSD. Thus, you are encouraged to consider how your efforts to research or apply

[3] Efforts like the FLOSSmole repository (www.flossmole.org) and the SourceForge.net Research Data repository (www.nd.edu/~oss/Data/data.html) are two emerging examples in this area.

software process concepts, techniques, or tools can be advanced through studies that examine processes found in F/OSSD projects, and that practice free or open source sharing, reuse, and extension of *software process* data, artifacts, models, and public repositories.

Acknowledgments

The research described in this report is supported by grants from the U.S. National Science Foundation #0083075, #0205679, #0205724, and #0350754; and from the EU to the CALIBRE project. No endorsement implied.

References

Antoniades, I.P., Samoladas, I., Stamelos, I., Angelis, L., and Bleris, G.L. 2004. 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.

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.

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

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 2003.

Elliott, M. and Scacchi, W., 2004. 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.

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 2002.

Feller, J., Fitgerald, B. Hissam, S. and Lakhani, K. 2005. *Perspectives on Free and Open Source Software*, MIT Press, Cambridge, MA.

Gacek, C. and Arief, B. 2004. The Many Meanings of Open Source, *IEEE Software*, 21(1), 34-40, January/February.

Gay, J. (Ed.). 2002. *Free Software, Free Society: Essays by Richard M. Stallman*, Free Software Foundation, Cambridge, MA.

Ghosh, R. and Prakash, V.V. 2000. The Orbiten Free Software Survey, *First Monday*,

5(7), July. Also see http://www.infonomics.nl/FLOSS/ for further information.

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 2002.

Hars, A. and Ou, S. 2002. Working for Free? Motivations for participating in open source projects, *Intern. J. Electronic Commerce*, 6(3), 25-39.

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.

Highsmith, J. 2002. *Agile Software Development Ecosystems*, Addison-Wesley, Boston, MA.

Hissam, S., and Plakosh, D. 1999. *COTS in the Real World: A Case Study in Risk Discovery and Repair*, (CMU/SEI-99-TN-003) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://www.sei.cmu.edu/publications/documents/99.reports/99tn003/99tn003abstract.html

Huntley, C.L. 2003. Organizational Learning in Open-Source Software Projects: An Analysis of Debugging Data, *IEEE Trans. Engineering Management*, 50(4), 485-493.

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 2004.

Jensen, C. and Scacchi, W. 2005. Process Modeling Across the Web Information Infrastructure, *Software Process—Improvement and Practice*, 10(3), 255-272, July-September.

Kim, A.J. 2000. *Community-Building on the Web: Secret Strategies for Successful Online Communities*, Peachpit Press, Berkeley, CA.

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

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. 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.

Madey, G., Freeh, V., and Tynan, R. 2004. Modeling the F/OSS Community: A Quantative 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.

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., Flanagin, 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. 2001. Specifying Process-Oriented Hypertext for Organizational Computing, *J. Network and Computer Applications,* 24(1), 39-61.

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

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.

Pavelicek, R. 2000. *Embracing Insanity: Open Source Software Development*, SAMS Publishing, Indianapolis, IN.

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.

Preece, J. 2000. *Online Communities: Designing Usability, Supporting Sociability*. Chichester, UK: John Wiley & Sons.

Raymond, E.S. 2001. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly Media, Sebastopol, CA.

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 2002.

Scacchi, W. 1999. Experience with Software Process Simulation and Modeling, *J. Systems and Software*, 46(2/3), 183-192.

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. Understanding Free/Open Source Software Evolution, in N.H. Madhavji, M.M. Lehman, J.F. Ramil and D. Perry (eds.), *Software Evolution and Feedback*, John Wiley and Sons Inc, New York, to appear, 2005a.

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, 2005b.

Scacchi, W., Jensen, C., Noll, J. and Elliott, M. 2005. Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes, *Proc. First Intern. Conf. Open Source Software*, Genova, Italy, 1-8, July 2005c.

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.

Sharma, S., Sugumaran, and Rajagopalan, B. 2002. A Framework for Creating Hybrid Open-Source Software Communities, *Information Systems J.*, 12(1), 7-25.

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 2004.

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.

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.

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

Ye, Y., Nakajoki, K., Yamamoto, Y., and Kishida, K., 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, 2004.

Ye, Y. & Kishida, K., 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 2003.

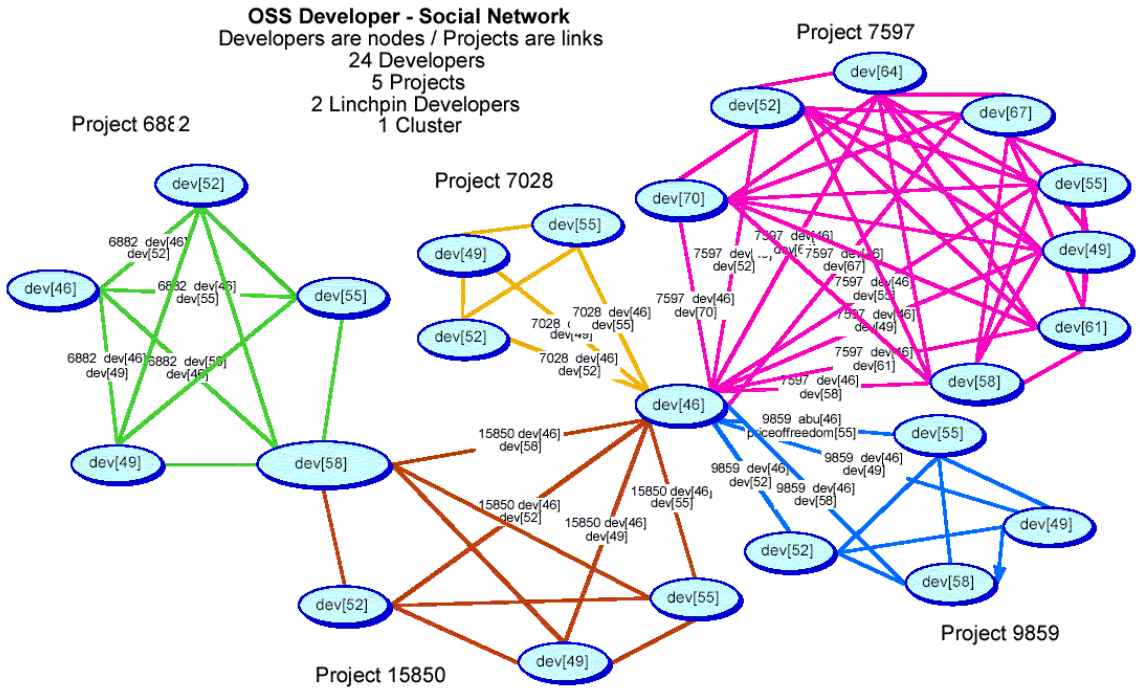


Figure 1. A social network that links 24 developers in five projects through two key developers into a larger F/OSS project community [cf. Madey 2004].

OSSD Processes Within or Across Communities

This section contains four chapters that document comparative studies of OSSD processes and practices found operating within or across different OSSD project communities. The first chapter examines and compares the world of OSSD and the world of academic software engineering. This chapter helps to reveal that OSSD and software engineering are quite independent approaches to the development of complex software systems in ways that are efficient, high quality, and cost effective.

The second chapter examines processes and practices for OSSD found in the world of networked computer game development. The computer game industry has now embraced and encouraged OSSD practices by users or consumers of their commercial closed source computer game engines and software development kits.

The third chapter examines the world of OSSD where beliefs about freedom and liberty are brought to bear in supporting the development of software for commercial business and electronic commerce (e-commerce) applications.

The last chapter provides an in-depth survey of the overall process of software evolution associated with OSSD projects. This was the first such study to find and report on the emerging pattern of sustained "exponential growth" of successful OSSD projects, as well as to develop some possible explanations based on multiple kinds of empirical evidence as to why such a pattern can emerge. The observed and independently substantiated pattern of exponential growth in successful OSSD projects, stands in contrast to the nearly 40 year long history of empirical software evolution studies that repeatedly found "inverse square" software growth curves characterizing the long-term evolution of conventional, closed source software developed within large corporate environments.

Walt Scacchi, When is Free/Open Source Software Development *Faster*, *Better*, and *Cheaper* than Software Engineering? Working Paper, Institute for Software Research, UC Irvine, April 2003.

Walt Scacchi, Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004.

Walt Scacchi, Open EC/B: Electronic Commerce and Free/Open Source Software Development, *Proc. 5th Workshop on Open Source Software Engineering*, 57-61, St. Louis, MO, May 2005.

Walt Scacchi, Understanding Free/Open Source Software Evolution, to appear in N.H. Madhavji, J.F. Ramil, M.M. Lehman, and D. Perry (eds.),

Software Evolution and Feedback, John Wiley and Sons Inc, New York, 2006.

When Is Free/Open Source Software Development *Faster*, *Better*, and *Cheaper* than Software Engineering?

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425 USA
+1-949-824-4130 (voice)
+1-949-824-1715 (fax)
Wscacchi@uci.edu

June2004

Acknowledgements: The research described in this report is supported by grants from the National Science Foundation #ITR-0083075, #ITR-0205679 and #ITR-0205724. No endorsement implied. Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at the Santa Clara University; Margaret Elliott, Chris Jensen, Mark Bergman, and Xiaobin Li at the UCI Institute for Software Research, and also Julia Watson at The Ohio State University are collaborators on the research project that produced this chapter.

When Is Free/Open Source Software Development *Faster*, *Better*, and *Cheaper* than Software Engineering?

ABSTRACT

This chapter draws attention to the question of determining the conditions when free/open source software development may represent a significant alternative to modern software engineering techniques for developing large-scale software systems. F/OSSD often entails shorter development times that can produce higher quality systems, and incur lower costs than may be realized through developing systems according SE techniques. Understanding why and how this may arise is the focus of this chapter. It presents, analyzes, and compares data collected from different F/OSSD projects, including an in-depth case study, to help develop such an understanding. The goal of this chapter is to determine the circumstances and conditions when F/OSSD represents a viable alternative to SE for the development of complex software systems. In particular, the chapter seeks to contrast differences observed in the arrangement and tooling of their respective software development practices, production resources, technical regimes, and community practices in which they are embedded. This in turn may then help identify how the practice and principles of SE might be improved.

Keywords

Software Engineering, Free/Open Source Software Development, Software Productivity, Software Quality, Software Cost

1.Introduction

Software engineering (SE) and free/open source software development (F/OSSD) are different approaches to the challenge of developing, deploying, and sustaining complex software system

products or services. Some have asserted that F/OSSD represents a significant alternative [DiBona *et al*., 2000, Pavlicek 2000] to the development of software commodity products or application services [cf. Wheelwright and Clark 1994]. Whether F/OSSD represents a quicker, more effective, and lower cost approach than SE, and under what circumstances and conditions, thus merits serious review. Similarly, the popular mantra of "faster, better, cheaper" suggests that new approaches to engineering, product development and innovation may be at hand and available [McCurdy 2001, Voas 2001, Wheelwright and Clark 1994]. However, little is known about how people in F/OSSD communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success. Such conditions may point to the need to critically reflect on whether the practices and principles of SE require a serious rethinking and possible reformulation to address and accommodate F/OSSD, as well as how F/OSSD differs from current SE principles. To the extent that academic communities, commercial enterprises, or government agencies seek the supposed efficacy of F/OSS, they will need grounded models of the processes and practices of F/OSSD to allow effective investment of their limited resources.

If it is true that F/OSSD is faster, better, and cheaper than SE under certain conditions, then is it possible to see if similar conditions could improve the practices and adapt the principles of SE? Has F/OSSD demonstrated the practical value and success of informal approaches, compared to the formal notation-based approaches advocated by SE scholars? Questions like these cannot be ignored or slighted by mere reference to more than three decades of academic and industrial SE research. Instead, this chapter brings questions like these into the foreground so as to advocate the position that the SE community needs to recognize how, and under what conditions, F/OSSD may represent a faster, better, and cheaper alternative for how to engineer complex software

systems. Failure of the SE community to embrace F/OSSD as something different than current SE principles, may relegate the future of SE research to that of an academic curiosity, rather than as an engineering discipline whose capabilities are maximized when operationalized as a complex web of socio-technical development processes and community oriented work practices.

2.Modern software engineering principles and best practices

In order to determine if and when F/OSSD outperforms modern SE, it is reasonable to first identify what principles and best practices of SE are being addressed. Clearly, there is no fixed or prior technical boundary that separates SE from F/OSSD, since software developers may or may not be free to select the methods, techniques, tools, and development processes that makes their work comprehensible and manageable. Instead, SE and F/OSSD may simply represent two alternative approaches to address the same challenge, which is developing large software system products or application services in an efficient, quality-oriented, and cost effective manner. Other alternatives include agile software development [Cockburn 2002, Fowler 2003] and extreme programming [Beck 1999]. However, understanding when and how a particular approach like F/OSSD may outperform SE is the focus here.

SE is an academic discipline and industrial practice that seeks to rationalize the development of complex software system products and services. It first appeared in the late 1960's, and its principles have been identified, captured, and increasingly taught as a subject suitable for late undergraduate or early graduate study [Brooks 1995, Pressman 2001, Schach 2002, Sommerville 2001]. A quick review of the most current edition of the textbooks cited finds the following principles: First, SE is a team endeavor that is focused on the development of large software systems through a software development life cycle. Second, the software life cycle (model) constitutes a framework that stipulates or orders the processes of SE that every software

development project should traverse. Third, the focal processes of SE include software requirements engineering, specification and prototyping, design (functional, architectural, modular, or object-oriented), testing (verification and validation), configuration management, maintenance (or evolution), and project management. Fourth, these processes may or should employ formal notations and reasoning schemes for consistency and completeness, though which computer-based tool to use to support such notations and schemes is unclear. Fifth, software quality results from the systematic performance of software life cycle processes that create, reuse, manipulate, or update software artifacts (including formal notations, graphic diagrams, and source code), according to project planning, cost estimation, and management control efforts. Sixth, the level, goal, or threshold of software quality (e.g., end-user satisfaction, number of defects discovered post delivery) that is sought or attained determines the level of software productivity that is achieved, as well as the overall cost of the software development effort. Following these principles often leads to product development and release cycles that are measured in months to years of calendar time and staff effort.

When these principles of modern SE get applied in industrial centers or in government system acquisition programs, a number of lessons learned emerge which are generally recognized as "best practices" for developing software system products or services through SE. A sample of best practices appears in Exhibit 1 as items grouped according to whether they address project integrity through project management, software construction integrity, and software product stability. These practices are drawn from the Web site of the Software Program Managers Network[1]. These practices draw attention to risk management, project performance metrics, defect tracking, and testing as a continuous ongoing process, within centrally located and

[1] SPMN was originally established by the U.S. Navy to capture, identify, and disseminate best practices for developing large software systems acquired by the U.S. Department of Defense. The SPMN Web site is maintained by Integrated Computer Engineering, Inc.

hierarchically organized corporate centers for software production, as supplements to the SE principles already identified above. Together, these principles and practices characterize the technical regime[Nelson and Winter 1982] for developing software products and service through SE.

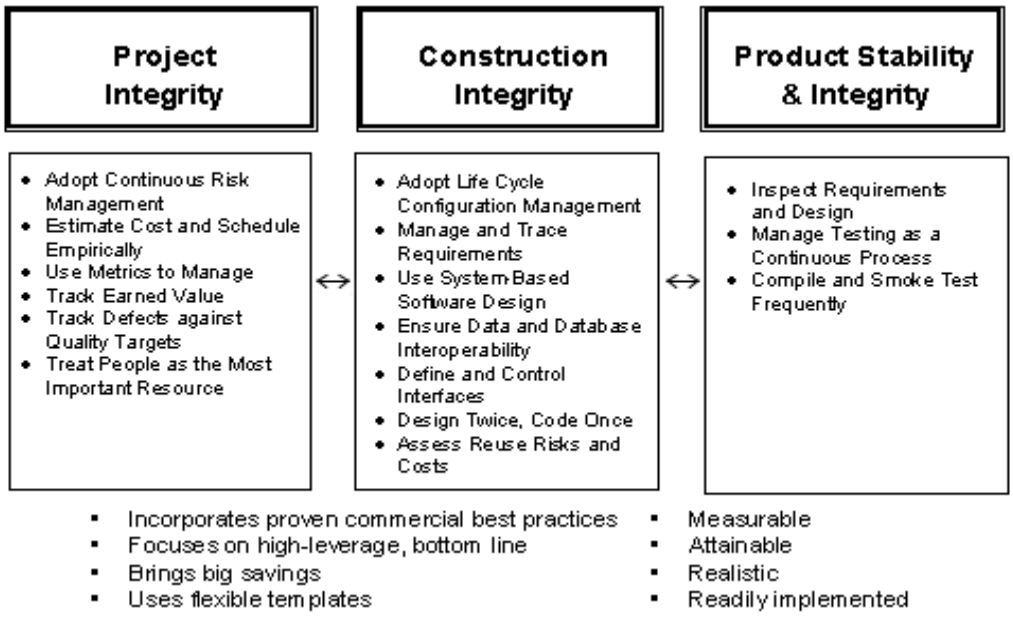


Exhibit 1. A set of best practices for software engineering project management (source: **http://www.spmn.com**, Copyright © 2003, Integrated Computer Engineering, Inc.).

Finally, with the encouragement and support of the U.S. Department of Defense, industry-wide efforts to improve software product quality have been promoted that entail the external assessment of the "maturity" of a firm's software development capability [Stalk and Hout 1990] that can be observed or measured in terms of the SE principles and best practices that it employs on a routine basis. However, whether or how the assessment of a firm's capability maturity does in fact constitute a reliable indicator or predictor of the quality of the software that is produced is unclear [Beechman 2003, Conradi 2002].

3.Do those who advocate F/OSSD practice modern SE principles?

With the foundation of principles and best practices for modern SE at hand, attention can now turn to examine the practices of F/OSSD. No effort is made to identify the principles of F/OSSD, since principles often take years of practice, empirical observation, conceptualization, abstraction, and refinement to identify, reproduce and stabilize, whereas the practice and technical regime of F/OSSD as a widespread approach to software development is about 10 years old, and systematic empirical studies have only begun to appear in the last few years.

There are at least four different places to examine to find what the practices of F/OSSD are. First, some F/OSSD projects seek to embrace modern SE principles, but may do so through practices different to those found in industry best practices noted above. An example here can be found in the dozens of F/OSSD projects associated with the Tigris.org OSSE community. Exhibit 2 presents the best practices they have identified.

Second, there are F/OSSD projects that are supported by, or organized within, industrial software development centers. Examples here include the NetBeans and Eclipse OSSD projects that are both developing Java-based interactive development environments (IDEs), based in part on the corporate support respectively from SUN and IBM.

Third, there are the vendors of OSSD project management environments like SourceForge Enterprise Edition™ from VA Software, SourceCast™ from Collab.Net, and Corporate Source from Zee Source. SFEE, SC, and CS are the products of early commercially oriented OSSD projects that have been evolved and refined into Web-based project management environments for collaborative software development [Augustin 2002]. These environments are not IDEs like

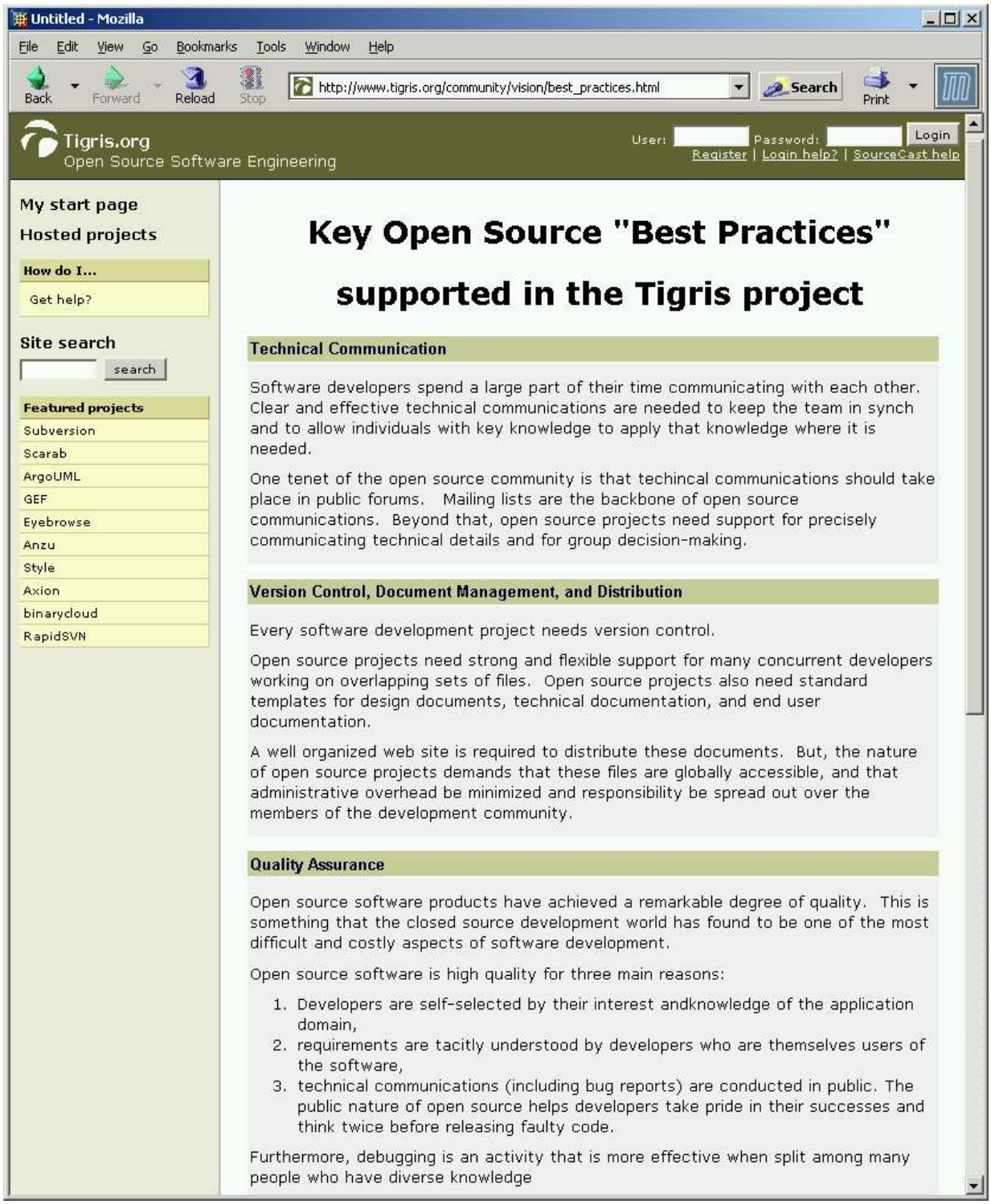


Exhibit 2. A partial view of best practices advocated for Tigris.org projects (source: **http://www.tigris.org/community/vision/best_practices.html**, April 2003).

NetBeans or Eclipse, though they could be made to interoperate with them. However, these environments are non-free commercial products or service offerings marketed primarily to large corporations that may have dozens or hundreds of organizationally dispersed software development projects underway at any one time. Companies like Hewlett-Packard, Barclays Global Investments, and others have adopted OSSD project management environments for use behind the corporate firewall [Dinkelacker 2002], or to support corporate sponsored OSSD projects like NetBeans. These OSSD projects follow practices that arise from the ongoing, routine use of the tools, services, and transactions supported within the project management environments for collaborative software development that these vendors offer. An example view of the project management activities, services or capabilities that are supported by SFEE, SC and CS appear in Exhibit 3. However, it should be noted that most F/OSSD projects do not employ all of these capabilities, though projects do [Halloran 2002, Scacchi 2002a].

Product Development	**Technical Communications**	**Project Management**	**Project Management**
Web-based source code access	Web-based file and content management	Incremental or partial project planning	Project/task status tracking
Bug and issue-tracking	Mailing list management	Process/workflow support	Update tracking
Configuration and version mgmt.	Discussion forums	Role-based access control	Audit logs and history
Search/index across source code and documents	Project document (Web page) templates	Enterprise or project branding	

Exhibit 3. Overall set of software product development, technical communication, and project management capabilities available in commercial OS collaborative software environments (sources: VASoftware SFEE, Collab.Net SC, and ZeeSource CS, April 2003).

Fourth, there are empirical studies that collect and analyze software development practices and processes within or across different samples of F/OSSD projects. These studies produce quantitative results that characterize F/OSS properties (source size, team size, release rates, bug/defect rates, etc.) or qualitative studies that identify processes, project ethnographies, or patterns of recurring activity for F/OSS development and evolution. Results from these studies will be presented later, though the following sub-section serves as an example of such a study.

3.1Case study: *Tigris.org and ArgoUML.tigris.org*

Consider the following case of the Tigris.org F/OSSD community, and one of the F/OSSD and SE projects affiliated with it, ArgoUML. Tigris.org operates like a virtual enterprise for decentralized software development [Noll 1999]. It exists primarily as a Web portal that operates on a SourceCast project management environment from Collab.Net. Thus it may encourage development practices similar to those of internal corporate or external corporate-sponsored OSSD projects. The ArgoUML project is an OSSE project based at argouml.tigris.org that focuses on the development of a computer-aided tool for developing software system designs notated in the UML. The home page for ArgoUML is displayed in Exhibit 4. No observations that follow are intended to denigrate or accentuate the important and valued efforts of this community. Instead, the purpose is to provide a real-world example of what happens when F/OSSD and SE come together.

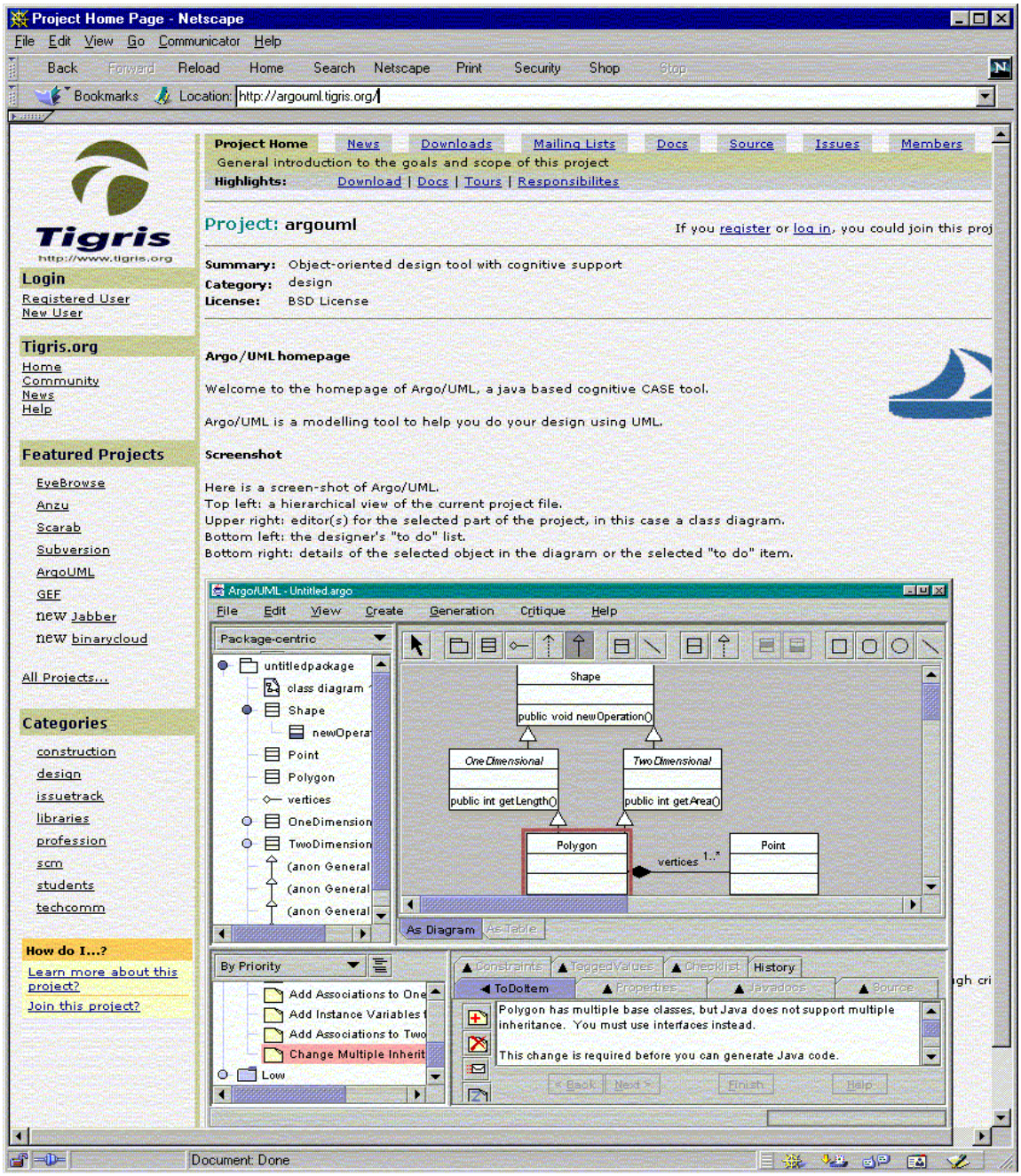


Exhibit 4. The homepage of the ArgoUML project on the Web (source: http://argouml.tigris.org)

The Tigris.org community identifies itself on its Web site portal as being a meeting ground for OSS developers and SE specialists and students. In browsing the Web site for Tigris.org, one finds in its Mission Statement:

> *"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."* (http:www.tigris.org, March 2002).

Such a claim might therefore lead one to expect to find numerous examples and instances of modern SE techniques and concepts being applied to support F/OSSD. For example, F/OSSD seems to focus attention to source code development and debugging [DiBona 1999, Pavilcek 2000]. Thus modern coding techniques like modularity and the use of program debugging and execution monitoring tools are expected.

Beyond this, most SE textbooks draw attention to topics like requirements engineering, software architecture and component design, validating an implementation (i.e., source code) satisfies its requirements, while testing/verifying the implementation is a consistent, complete, traceable, and in some way correct realization of its architecture and component design. Project management and configuration management also receives appropriate attention. ArgoUML seeks to embrace OSSD and SE principles through creation of:

> "*…a modeling tool to help you do your design using UML…and it is also an Open Source Development project where you are invited to contribute*". ArgoUML is also "*…a domain-oriented design environment that provides cognitive support of object-oriented design*" (http://www.argouml.tigris.org, March 2002).

The ArgoUML project today includes more than 19,000 registered users and over 150 developers. ArgoUML is thus a large software development project with a significant number of users that is conceived to support SE professionals using modern SE design tools and techniques. UML is a widely recognized unified modeling language that is the addressed in many SE and Information System Design textbooks, and found in use in many industrial SE R&D projects. Using UML, SE/ISD professionals can create or document Use Cases for software requirements. In addition, UML is a notation for specifying software component design and architectural features of component arrangements. However, nowhere on the ArgoUML Web site can one find any Use Case diagrams that specify the requirements for ArgoUML, nor any UML descriptions of ArgoUML's architecture or component design. Thus, it appears that ArgoUML developers' do not practice using the tool itself to document its own development. As such, perhaps it's not surprising to discover:

> *"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."*
> (http://www.argouml.tigris.org, March 2002).

So what are SE professionals suppose to learn from the ArgoUML experience in OSSE? Is SE good for someone else, or for students to study, but not for those who actually build SE tools that support modern SE techniques and concepts? Similarly, in examining any of the remaining 35 or so other projects affiliated with Tigris.org, it is difficult to find what SE tools, which are being developed within the Tigris.org community, actually are being used by other projects within the community[2], and whether any were engineered using SE techniques like Use Cases for requirements and UML for their design. Instead, the situation we find is better characterized as:

[2] The configuration management tool, *Subversion*, is being use to manage its own source code configuration. In contrast, it is unclear whether the issue tracking (or bug reporting) system, *Scarab*, is being used to track issues arising during its development, or in the development of other Tigris.org projects. This observation is not intended to be in any way a positive/negative assessment of these F/OSSD projects, but merely to highlight that F/OSSD and SE practices are different.

> *"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 great for developers to keep their skills current and plug into a growing base of shared experience for everyone in the field."* (http://www.argouml.tigris.org, March 2002).

In this case of the Tigris.org community and ArgoUML project, but not generalizing to all OSSE efforts, it appears that the objectives, practices and technical regimes of F/OSSD and SE are different. However, as a note of caution, these results also should help researchers investigating F/OSSD projects recognize the potential risks for making pre-mature generalizations about typifying what F/OSSD is, or how it works, based on the examination of a single F/OSSD project, or even a single F/OSSD community [cf. Scacchi 2002a]. What is true of one F/OSSD project's artifacts, processes, or practices may not be true of any other F/OSSD project, without empirical study and explicit comparison.

With this modest grounding of exhibits and case study of F/OSSD efforts that in some way address SE topics or concerns, it is possible to examine the overarching question which this chapter addresses. Note that the opening question does not focus on attributes of F/OSS source code programs or other executable implementations (e.g., make files, operating system shell scripts, plug-in modules (like "ActiveX controls"), or intra-application scripting code like JavaScript). Instead focus is directed to attributes of F/OSSD processes, technical regime, management and work practices, and collective community-sustaining actions within projects.

4.How is F/OSSD *faster* than SE?

What does it mean for one software product development approach to be “faster” than another? Space/time measures like speed come to mind first, but for software development the most

common indicators include software productivity (e.g., source lines of code or product feature delivered per developer work time unit [Sommerville 2000]), task duration schedules (process cycle time [Stalk and Hout 1990]), product delivery time (e.g., time to market, time to next product innovation [Kogut and Metiu 2001, von Hippel 2001]), and product defect repair time.

Large composite F/OSSD projects, like those at NetBeans.org, Apache.org, Mozilla.org, Tigris.org, Debian.org, the Linux Kernel, or the corporate users of SFEE, SC or CS, seek to enact "Internet time" development practices, much like Microsoft, Netscape, and others [Cusomano 1999, MacCormack 2001]. Internet time software development projects emphasize minimizing time to market and delivery of incremental improvements (e.g., user initiated innovations) in functionality, instead of complete well-engineered products. Incremental product releases are driven by feedback from users as a way to determine which incremental functionality and which perceived errors in available functionality matter most, as well as how they might be improved or resolved [DiBona 1999, Dinkelacker 2002, Pavlicek 2000]. Internet time and F/OSSD projects also tend to produce incremental software releases at a much faster rate, even to the point of releasing unstable but operational daily system builds. This denotes not only a reduction in product release cycle times compared to SE practice, but also a significantly restructured life cycle process and process cycle time reduction.

Many of the largest and most popular F/OSS systems like the Linux Kernel [Godfrey 2000, Schach 2002b], GNU/Linux distributions [O'Mahony 2003], GNOME user interface [Koch 2002] and others are growing at an *exponential* rate, as is their internal architectural complexity [Schach 2002b]. F/OSS system architectures and functionality can grow in discontinuous jumps as independent F/OSSD projects merge, as their autonomously designed and evolved systems are

combined [Nakajoki 2002, Scacchi 2002b]. F/OSSD project teams also continue to grow over time and across releases at an incremental rate, suggesting that adding developers "late" in its development cycle may not slow it down, but may instead increase the size, functionality, and quality of the system. This stands in contrast to the long received wisdom of SE that indicates that adding developers to a project soon before release, delays the release and has an adverse effect on system quality [Brooks 1995]. Thus, there are examples of where F/OSSD projects are producing large software systems whose size and productivity grows at a rate faster than observed for SE projects that focus attention to product development scheduled and management control [cf. Lehman 2002].

As F/OSS developers are themselves often end-users of their systems, then software requirements and design take less time to articulate and negotiate, compared to SE projects. For example, Exhibit 1 identifies the SE practice of organized inspection of (explicit) software system requirements and design artifacts, while Exhibit 2 identifies the practice (see Quality Assurance item 2) of software requirements and designs that are not typically written down or formalized [Scacchi 2002a], but are nonetheless tacitly understood by developers and users. Similarly, in SE, developers are not expected to be users of the systems they develop. As such, they must elicit requirements and validate system design with end-users who are generally not SE professionals, and thus must negotiate what they will be able to do, on what schedule and budget, and with what staff resources. In contrast, when F/OSSD projects involve users as developers, the time it takes to determine required system functionality is shorter, and often less demanding that expected in SE projects. Thus, the elapsed time intervals for incremental software product innovation, defect detection and removal, and product adaptation and release cycles are shorter, while overall growth in software product functionality and size is faster.

5.How is F/OSSD *better* than SE?

What does it mean for one software product development approach to be "better" than another? Measures or indicators of product quality (e.g., product defects discovered in the field per release) come to mind first. However, other indicators of increased effectiveness of software development include product reliability and security [Sommerville 2000], response time to diagnose and repair product defects [Mockus 2002, Zhao 2003], as well as increases in social welfare in the product developer or user community associated with ease and openness of effective technical communication [Yamauchi 2000], building sustained trust [Pavlicek 2000], accumulation of social capital by developers and user-contributors [Berquist 2001], and community ownership and protection of shared/common pool resources [cf. Ostrom. 1994].

F/OSSD projects rely on software *informalisms* [Scacchi 2002a] as shared information artifacts (resources) that can be publicly accessed, browsed, hyperlinked, and updated on demand. These informalisms, like threaded email discussion lists and project Web pages, are socially lightweight mechanisms for managing, communicating, and coordinating globally dispersed knowledge about who did what, why, and how [Mackenzie 2002, Sharman 2002, Yamauchi 2000]. These informalisms are easy to learn and use as semi-structured representations that capture software requirements, system design, and design rationale, though they are not often identified as such. Large OSS systems, like the Apache Web server and Mozilla Web browser, that are developed and sustained through informalisms have been found to have more stable features of higher quality compared to systems developed using traditional SE techniques within corporate settings [Dinkelacker*, et al*, 2002, Mockus, *et al*., 2002, Zhao and Elbaum 2003]. The informalisms used in F/OSSD projects stand in contrast to the more cumbersome, more precise and more

demanding heavyweight formalisms advocated for use, following the principles and practices of SE.

F/OSSD projects are iteratively developed, incrementally released, reviewed and refined by software development peers in an ongoing agile manner [cf. Cockburn 2002, Fowler 2003, Kogut and Metiu 2001]. These methods ensure adaptation to shifting user/developer requirements that are conveyed through informalisms. They also ensure acceptable levels of quality, coherence, and security of system-wide software via continuous distributed testing and profiling [Payne 2002, Schmidt 2001]. Agile software development practices are therefore closely aligned to F/OSSD practices, though it may be fairer to observe that agile software development methods stand somewhere in the middle ground between SE and F/OSSD practices.

F/OSSD projects are hosted within decentralized communities of peers [Kogut 2001, O'Mahony 2003, Scacchi 2002a, 2002b, Sharman 2002] that can form a virtual enterprise interconnected via logically centralized Web sites and repositories [Noll 1999]. F/OSSD projects rely on their software developers to provide peer reviews of source code and system quality that are synchronized with the complexity of the system release. Major releases undergo more review compared to daily build releases. However, peer review helps create a community of peers, which is one way how social capital may be accumulated (by contributing well regarded review discussions) or lost (by initiating a "flame" discussion that distracts or offends other project contributors and thus discredits the initiator). Community oriented F/OSSD also gives rise to new kinds of requirements for community building, community portals (e.g., SourceForge.net, Freshmeat.net, Savannah.gnu.org, Tigris.org), community software, and community information sharing systems for Web site content management and interlinked communication channels for

email, forums, and instant messaging [Scacchi 2002a]. These convivial capabilities and community-oriented mechanisms tend to improve the social welfare of the contributing F/OSSD developers and users. In contrast, most SE projects are targeted for hosting within a centralized corporate setting, where access and visibility may be restricted to local participants under the administrative control of project or business managers. But F/OSS systems also co-evolve with the community of developer-users who create and contribute to them [Nakajoki 2002, O'Mahony 2003, Scacchi 2002a]. However communities, as a form of social organization for software development and for learning about software technology practices, are not mentioned in modern SE principles, practices or textbooks.

The vast majority of F/OSS projects are small, short-lived, exhibit little/no growth, and often only involve the effort of one developer [Hunt 2002, Madey 2002]. In contrast, a few large projects realize a critical mass of 5-15 core F/OSS developers [Madey 2002, Mockus 2002] and inevitably garner the most attention, software downloads, and usage. But what is significant about this overall population of projects and developers is that as many as 60% or more F/OSS developers participate in two or more projects, and more than 5% participate in 10 or more F/OSS projects [FLOSS 2002, Hars 2002]. These have been labeled as "linchpin developers" [Madey 2002] to indicate their role in enabling previously independent small F/OSS projects. These developers come together as a larger social network with the critical mass needed for their independent systems to be merged and experience more growth in size, functionality, and user base. Whether such a trend is found in traditional SE projects is unclear.

Finally, it appears that F/OSSD projects rely on *virtual project management* to mobilize, coordinate, control, build, and assure the quality of F/OSSD activities. VPM invites or

encourages system contributors to come forward and take a shared, individual responsibility that will serve to benefit the F/OSSD collective of user-developers. VPM requires multiple people to come forward to act in the role 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 community development [Fielding 1999]. In contrast, SE projects are predicated on centralized project management regimes where project managers are assigned the administrative authority to plan, manage, and control software development resources, staff, schedule, and budget their projects. F/OSSD projects enable participative management that can arise from a decentralized community of developers, whereas SE assumes delegative management that arises from a centralized corporate structure and resource control framework.

6.How is F/OSSD *cheaper* than SE?

What does it mean for one software product development approach to be "cheaper" than another? Measures or indicators of monetary cost like project budgets [Sommerville 2000] come to mind first, but other costs merit and garner more attention. These may include procurement (purchase) and deployment costs, tooling or production infrastructure costs, transaction costs associated with the governance or coordination of development activities, opportunity costs (e.g., costs associated with not choosing one option over another), and cost transfers (e.g., subsidies).

F/OSSD tools and application systems are inexpensive/free to acquire, comparatively easy to use and learn, and are globally accessed and transferred across the Internet [DiBona 1999, Pavlicek 2000]. In many situations, there are competing alternative implementations of F/OSS tools, so that developers can assess and evaluate which one best meet their need or taste. These tools are both given and received as public goods or gifts [Bergquist 2001]. F/OSS operating environments like the Debian GNU/Linux distribution constitute thousands of utilities, tools and

end-user applications, whose overall development costs is estimated in billions of dollars [Gonzalez-Barahona 2003, O'Mahony 2003], are available as F/OSS for immediate download from the Internet. Substantial F/OSSD tool collections [Halloran 2002] are essentially free, therefore have fewer cost barriers to their procurement and adoption, and are more readily transferred within and across communities of developers and users. Commercially available SE tools like Rational Rose™ or Microsoft Visual Studio.Net™ are not free, though they may be available for free trail periods of days or weeks. However, they may have barriers to their adoption in form of perceived higher costs in their acquisition, training, number of user licenses, and product support. Commercially available collaborative software development environments like SFEE, SC and CS, though also not free, are packaged for sale with support services that similarly represent the major cost of their procurement and sustained use.

F/OSS tools and applications are among the most widely used examples that demonstrate the potential for software reuse [Brown 2002]. Successful and widespread reuse saves time and reduces development costs by avoiding redevelopment of previously developed products, components or modules. However, F/OSSD encourages not only software reuse and resource sharing, but in many ways encourages the ongoing evolution of tools and applications through reinvention as a basis for continuous improvement. Faster and better F/OSSD conditions in turn tend to drive down the cost of developing software, at least in terms of schedule and budget resources. SE of course also encourages software reuse through advocacy of component-based system design and use of commercial-off-the-shelf (COTS) components. However, it may be the situation that SE encourages invention over reinvention, and relies primarily on corporate initiatives for software process improvement, but with mixed results [Beechman 2003, Conradi

2002], instead of developer-user motivation for software product improvement [Hars 2002, von Hippel 2001].

Most F/OSSD projects are voluntarily staffed by developer-users who want to work on the project, who will potentially commit their own time, skill, effort, and personal computing resources, thereby subsidizing or reducing the apparent cost of F/OSSD. In exchange, these contributors may realize personal, professional, or "private-collective" benefits from the F/OSSD development efforts [FLOSS 2002, Hars 2002, von Hippel and von Grogh 2003]. Minimal management or governance forms [Fielding 1999, Sharman 2002] are used to direct F/OSSD efforts, compared to the more rigidly hierarchical, managed, planned, staffed, controlled, and budgeted project activities typical for SE best practice efforts.

7.Discussion

In contrast to large, sustained F/OSSD projects, SE embraces a rational economic approach to the private development of complex software systems or applications as commodity products or proprietary services. F/OSSD project communities on the other hand appear to be motivated by collective actions that create, utilize, extend, and redistribute common pools of software resources and information artifacts (programs in source code and executable forms, informalisms, etc.). Thus, this is perhaps another reiteration of the classic institutional conflict between rational action of private capital/firms versus the collective action associated with F/OSSD of a community that seeks to manage and share its common-pool resources [Hayek 1945, Ostron 1994, North 1990]. Consequently, resolution of such a conflict may therefore lie somewhere in between, as suggested by the private-collective mode of innovation and software production [von Hippel and von Grogh 2003].

Large, globally dispersed F/OSSD projects enact lateral organizational relations [Galbraith 1996] among their developers and users through meritocratic teamwork structures and peer-oriented decentralized community forms. These relations reduce or supplant hierarchical functional organizational forms inherent in traditional SE techniques that increase bureaucratic tendencies through rules and formalization [cf. Ostrom 1994]. F/OSSD relies on the private-collective actions of developers and users to realize virtual project management capabilities that reduce reliance on formal project management techniques and administrative structures that pervade industrial SE projects, and that are reiterated in SE textbooks. F/OSSD is oriented to community and agility, rather than oriented to centralized project management and formal life cycle documentation regimes. Developers as users reduces the time and effort needed to figure out what users want, and whether what is developed and delivered meets user needs [von Hippel 2001, von Hippel and von Grogh 2003]. Thus, drawing on the practices observed in F/OSSD projects, the opportunity exists for developing new SE processes, practices, project community forms or organizational architectures [Nadler and Tushman 1997] that are decentralized, peer-oriented (lateral), and rely on semi-structured, informal representations of software artifacts. SE community Web sites that host OSSE examples and community development tools also appear to be candidates for adoption.

F/OSSD is not a panacea compared to SE, nor is it without its shortcomings. As noted, the vast majority of F/OSSD projects fail to grow beyond 1-2 developers, and subsequently their associated software source code never achieves a critical mass of users, functionality, community discourse informalisms or related resource subsidies. So F/OSSD in general is a risky undertaking, at least in terms of the probability of achieving critical mass, as well as realizing a faster, better, and cheaper way to develop complex software products or services. Accordingly,

F/OSSD is not well suited for adoption in hierarchical organizations that develop software products or services through rational management schemes traditional to SE principles and practices [cf. Dinkelacker 2002]. Organizations or firms that develop software products or service that are not wedded to the tradition of SE may on the other hand, find the adoption of F/OSSD practices as a viable alternative, even when developing high-value software products like enterprise resource planning (ERP) systems.[3] Even large software development companies like IBM, SUN, HP, and SAP have all started or sponsored F/OSSD projects (IBM Eclipse, SUN NetBeans, HP Gelato, SAP DB-MySQL) that are separate from, but complementary to, their mainstream software product lines that tend to follow SE practices.

Beyond this, the overall risk of failure in F/OSSD can be reduced or mitigated by association (social networking) with other complementary F/OSSD projects, as suggested by the Tigris.org community example, other studies [Hars 2002, Madey 2002], or the corporate F/OSSD efforts noted above. Such associations, if successful and sustained, enable the transactional flow of shared resources, gifts, trust, and social capital throughout the social and information networks of their developers and users [cf. North 1990]. So it appears that F/OSSD is something different than the rational economic world of SE, and thus merits further study, practice and refinement [cf. von Hippel and von Grogh 2003].

Last, those who teach SE should consider how to embrace F/OSSD practices, community information resources and infrastructure as an alternative approach to the development of large software system products or services. These practices build viable lateral social relationships that

[3]Compiere is a small software company that has developed an open source ERP system that operates in conjunction with Oracle's proprietary database management system. Compiere reports more than 400,000 copies of its ERP system have been downloaded, making it the most widely deployed ERP system in the world. Elsewhere, the GNUe.org community is also developing a "free software" only ERP system that is aimed at deployments in small companies and developing countries [Elliott and Scacchi 2003, Scacchi 2002b].

are cultivated, sustained, and evolved through the ongoing use, sharing, and reuse of F/OSSD informalisms. These informalisms encourage the sharing, study, reinvention, modification and redistribution of development project results. SE education need not be wedded only to the interests of rationally managed and hierarchically organized corporate software development projects. SE educators may also consider where and how to embrace a more free and open global community of students and collaborating software engineers who want to improve their software development productivity and quality, while reducing its costs. Finally, current SE textbooks are in need of revision to accommodate emerging F/OSSD principles and practices.

8.Conclusions

Free and open source software development appears to be changing the world of software development at a faster, better, and cheaper pace, and with a broad impact and audience. Understanding why this is so may help advance the state of the art of both SE and F/OSSD. Failing to recognize the differences between the two may result in F/OSSD characterizing more of the leading edge of global software product development activity, while SE characterizes more of the trailing edge of software development found in rationally managed corporate settings. Accordingly, this chapter examines the question of when is F/OSSD faster, better, and cheaper than SE through examination of data exhibits, a case study, and review of related empirical studies, and the results can be summarized as follows.

F/OSSD is faster than SE when the development life cycle is focused on Internet time product releases that can be produced and delivered on a daily basis. The rapid development and release cycle means that unstable but operational systems are delivered most of the time, while stable systems emerge slowly after extensive community review, iterative and incremental refinement, online discourse about system features and usage experience. Large F/OSS systems may grow in

size and platform diversity at a faster rate than systems resulting from SE, and these F/OSS systems are of acceptable quality to users. Independent F/OSSD projects can merge into larger projects which may then reach a critical mass of developers needed to obtain high growth rates in system functionality and quality. However, SE may be able to contribute expertise for how to restructure and reinvent the architecture of large F/OSS systems, so as to continuously improve and mitigate their unwanted complexity. Finally, most developers of F/OSS systems are themselves users of these same systems, thus they can more readily determine system requirements and design features through online discourse. All of these capabilities help make F/OSSD faster than traditional SE principles and practices.

F/OSSD is better than SE when developers rely on socially lightweight software informalisms rather than mathematically heavyweight software formalisms. F/OSSD projects that employ informalisms tend to be more agile, and the systems that result co-evolve with the teams and communities that develop them. F/OSSD projects rely on community building, community portals, community software, and community information sharing systems for project content management and communication channels to realize and sustain the quality in the software being developed. This socio-technical infrastructure for F/OSSD enables agile virtual enterprise forms that practice virtual project management through the routine use of collaborative software development tools, techniques, and informalisms.

F/OSSD is cheaper than SE when the total costs of F/OSS tools and end-users applications is low or free, though their collective development cost enables the flow of social capital among their developers and users. F/OSSD projects are a prime venue for demonstrating the value of software reuse and reinvention, and this speeds development and reduces its costs. Many F/OSS

developers appear motivated to give away the products of their collaborative software development work in order to share, examine, learn, reinvent, modify, and redistribute the results of their experiences and practice of F/OSSD in ways that build and sustain a community of like-minded developers.

Overall, F/OSSD is not an irrational version of SE, nor is it SE poorly done. Instead, F/OSSD is more of a private-collective approach to the problem of how to develop large software systems. F/OSS embraces, encourages, and perhaps requires more of a community oriented, collaborative software development effort as the basis for its practices and success. SE in turn can be made faster, better and cheaper by selectively adopting and integrating practices, technologies, and community techniques from F/OSSD projects. Empirical study of F/OSSD practices can therefore help identify new ways for how to improve the principles and practices of SE.

9.References

Augustin, L., D. Bressler, and G. Smith, 2002. Accelerating Software Development through Collaboration, *Proc. 24th. Intern. Conf. Software Engineering*, Orlando, FL, 559-563, May.

Beck, K., 1999. *Extreme Programming Explained: Embrace Change***,** Addison-Wesley Pub Co.

Beechman, S., T. Hall and A. Rainer, 2003. Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis, *Empirical Software Engineering*, 8(1), 7-42, March.

Bergquist, M. and J. Ljungberg, 2001. The power of gifts: organizing social relationships in open source communities, *Info. Systems J.*, 11(4), 305-320.

Brooks, F.P., 1995. *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition (2nd Edition), Addison-Wesley, Menlo Park, CA.

Brown, A.W. and G. Booch, 2002. Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors, *Proc. 7th Intern. Conf. Software Reuse*, Lecture Notes in Computer Science, Vol. 2319, 123-136.

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

Conradi, R. and A. Fuggetta, 2002. Improving software process improvement, *IEEE Software*, 19(4), 92-99, July-August.

Cusumano, M. and Yoffie, D.B. 1999. Software Development on Internet Time, *Computer*, 32 (10), 60-69, October.

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

Dinkelacker, J. P.K. Garg, R. Miller and D. Nelson, 2002. Progressive Open Source, *Proc. 24th Intern. Conf. Software Engineering*, Orlando, FL, 177-184, May..

Elliott, M., and W. Scacchi, 2003. Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, to appear in S. Koch (ed.), *Free/Open Source Software Development*, Idea Press.

Fielding, R.T. 1999. Shared Leadership in the Apache Project. *Communications ACM*, 42(4), 42-43.

FLOSS, 2002. *Free/Libre and Open Source Software: Survey and Study*, FLOSS Final Report, International Institute of Infonomics, University of Maastricht, The Netherlands, June.

Fowler, M., 2003. The New Methodology, http://martinfowler.com/articles/newMethodology.html, April.

Galbraith, J.R. 1994, *Competing with Flexible Lateral Organizations* (Second Edition), Addison-Wesley, Menlo Park, CA.

Godfrey, M.W. and Q. Tu, 2000. Evolution in Open Source Software: A Case Study, *Proc. 2000 Intern. Conf. Software Maintenance*, 131-142, San Jose, California, October.

Hars, A. and S. Ou, Working for Free? 2002. Motivations for Participating in Open Source Software Projects, *Intern. J. Electronic Commerce*, 6(3), 25-39.

Hayek, F.A. 1945. The Use of Knowledge in Society, *American Economic Review*, 35(4), 519-530.

Hunt, F. and P. Johnson, 2002. On the Pareto Distribution of SourceForge Projects, *Proc. Open Source Software Development Workshop*, 122-129, Newcastle upon Tyne, UK.

Halloran, T. and W. Scherlis, 2002. High Quality and Open Source Software Practices, *Proc. 2nd Workshop on Open Source Software Engineering*, Orlando, FL.

Koch, S. and G. Schneider, 2002. Effort, Co-operation, and Co-ordination in an Open Source Project: GNOME. *Info. Sys. J.,* 12, 27-42.

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

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.

MacCormack, A., R. Verganti, and M. Iansiti, 2001. Developing Products on Internet Time: The Anatomy of a Flexible Development Process, *Management Science*, 47(1), 133-150.

Mackenzie, A. P. Rouchey and M. Rouncefield, Rebel Code? 2002.The open source 'code' of work, *Proc. Open Source Software Development Workshop*, Newcastle upon Tyne, UK, 83-102.

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

McCurdy, H.E., 2001. *Faster, Better, Cheaper: Low-Cost Innovation in the U.S. Space Program*, John Hopkins University Press.

Mockus, A., R.T. Fielding, and J. Herbsleb, 2002. Two case studies of open source software development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, 11(3), 309–346, July.

Nadler, D.A. and Tushman, M.L. 1997. *Competing By Design: The Power of Organizational Architecture,* Oxford University Press, New York.

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

Nelson, R.R. and Winter, S.G., 1982. *An Evolutionary Theory of Economic Change*, Belknap Press, Cambridge, MA.

Noll, J. and W. Scacchi, 1999. Supporting Software Development in Virtual Enterprises, *J. Digital Information*, 1(4), February. http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll/.

North, D.C., 1990. *Institutions, Institutional Change and Economic Performance,* Cambridge University Press.

O'Mahony, S., 2003. Developing Community Software in a Commodity World, in M. Fisher and G. Downey (eds.), *Frontiers of capital: Ethnographic Reflections on the New Economy*, Social Science Research Council, (to appear).

Ostrom, E., Gardner, R., and Walker, J., 1994. *Rules, Games, and Common-Pool Resources*, University of Michigan Press, Ann Arbor, MI.

Pavlicek, R., 2000. *Embracing Insanity: Open Source Software Development*, SAMS Publishing, Indianapolis, IN.

Payne, C., On the Security of Open Source Software, *Info. Systems J.*, 12(1), 61-78, 2002.

Pressman, R., 2001. *Software Engineering: A Practitioner's Approach* (Fifth Edition), McGraw-Hill.

Scacchi, W., 2002a. Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings - Software*, 149(1), 24-39, February..

Scacchi, W., 2002b. *Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development*, Working Paper, Institute for Software Research, UC Irvine, July.

Schach, S.R., 2002. *Object-Oriented and Classical Software Engineering* (Fifth Edition), McGraw-Hill.

Schach, S.R. B. Jin, D.R. Wright, G.Z. Heller, and A.J. Offutt, 2002. Maintainability of the Linux Kernel, *IEE Proceedings – Software*, 149(1), 18-23, February.

Schmidt, D. and A. Porter, 2001. Leveraging Open-Source Communities to Improve the Quality & Performance of Open-Source Software, *1st Workshop on Open Source Software Engineering, Toronto, Ontario, May.*

Sharman, S., V. Sugurmaran, and B. Rajagopalan, 2002. A Framework for Creating Hybrid-Open Source Software Communities, *Info. Systems J.,* 12(1), 7-25.

Sommerville, I., 2000. *Software Engineering (6th Edition),* Addison-Wesley Pub Co.

Stalk, G. and T.M. Hout, 1990. *Competing Against Time: How Time-Based Competition is Reshaping Global Markets*, Free Press, New York.

Voas, J., 2001. Faster, Better, and Cheaper, *IEEE Software*, 18(3), 96-97, May-June.

Von Hippel, E., 2001. Innovation by User Communities: Learning from Open-Source Software, *Sloan Management Review*, 42(4), 82-86.

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.

Wheelwright, S.C. and Clark, K.B., 1992. *Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency, and Quality*, Free Press, New York.

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

Zhao, L. and Elbaum, S., 2003. Quality assurance under the open source development model, *Journal Systems and Software*, 66, 65-75.

Free and Open Source Development Practices in the Game Community

Walt Scacchi, *University of California, Irvine*

Empirical studies of four distinct free and open source software development communities find at least five common types of development processes. These communities, particularly the computer game community, provide examples of common practices.

The free and open source software (FOSS) approach lets communities of like-minded participants develop software systems and related artifacts that are shared freely instead of offered as closed-source commercial products. Free (as in freedom) software and open source software are closely related but slightly different approaches and licensing schemes for developing publicly shared software. Although the amount of popular literature that attests to FOSS is growing,[1,2] only a small—but growing—number of systematic empirical studies exist that explain how these communities produce software.[3–5] Similarly, little is known about how community participants coordinate software development across different settings, or about what software processes, work practices, and organizational contexts they need for success.

Academic communities, commercial enterprises, and government agencies that want to benefit from FOSS development will need grounded models of its processes and practices to effectively invest their limited resources. My team at the UC Institute for Software Research investigated the software development practices, social processes, technical system configurations, organizational contexts, and interrelationships that give rise to FOSS systems in different communities. In particular, we looked at the FOSS computer game community to provide examples of common development processes and practices.

Understanding FOSS development practices

No prior model or globally accepted framework exists that defines how FOSS is developed in practice. The starting point is to investigate FOSS practices in different communities.

Researchers are investigating at least four diverse FOSS communities through empirical

0740-7459/04/$20.00 © 2004 IEEE

FOSS development communities don't seem to readily adopt modern software engineering processes.

studies.[3,4,6,7] These communities center on software development for Web and Internet infrastructure, computer games, software engineering design systems, and X-ray and deep-space astronomy.

Rather than examining FOSS development practices for a single system (for example, GNU/Linux)—which might be interesting but is unrepresentative—or related systems from the same community (such as Internet infrastructure), my team's focus was to identify general FOSS practices both in and across these diverse communities. These practices were empirically observed in different projects from these communities using ethnographic methods detailed elsewhere.[6,7] Further, data exhibits in the form of screenshots from projects in the computer game community exemplify the practices. (On the SourceForge Web portal, computer games are the fourth most popular category of FOSS projects, with more than 8,000 out of the 70,000 total registered projects.) Comparable data from the other communities could serve equally well.

FOSS community participants often play different roles, such as core developer, module owner, code contributor, code repository administrator, reviewer, or end user. They contribute software content (programs, artifacts, execution scripts, code reviews, comments, and so on) to Web sites in each community and communicate their content updates via online discussion forums, threaded email messages, and newsgroup postings. Screenshots, how-to guides, and frequently asked questions also help convey system-use scenarios. Software bug reports appearing in newsgroup messages, on bug-reporting Web pages, or in bug databases describe what isn't working as expected. Administrators of these sites serve as gatekeepers by choosing what information to post, when and where on the site to post it, and whether to create a site map that constitutes a taxonomic information architecture for types of site- and project-specific content.

Software extension mechanisms and FOSS software copyright licenses that ensure freedom and openness are central to FOSS development. Extension mechanisms let people modify the software system's functionality or architecture via intra- or interapplication scripting or external module plug-in architectures. Copyright licenses, most often derived from the GNU General Public License, are attached to any project-developed software so that it can be further accessed, examined, debated, modified, and redistributed without future loss of these rights. These public software licenses contrast with the restricted access of closed-source software systems and licenses.

In each of these four communities, participants occasionally publish online manuals, technical articles, or scholarly research papers about their software development efforts,[1,3,8–10] which are then available for offline examination and review.

Each type content is publicly available data that can be collected, analyzed, and represented in narrative ethnographies, quantitative studies, or computational models of FOSS development processes. Significant examples of each kind of data have been collected, analyzed, and modeled.[3–5]

FOSS development processes

Unlike the software engineering world, FOSS development communities don't seem to readily adopt modern software engineering processes. FOSS communities develop software that's extremely valuable, generally reliable, globally distributed, made available for acquisition at little or no cost, and readily used in its associated community. So, what development processes are they routinely using and practicing?

From studies to date, they are employing at least five types of FOSS development processes. I'll briefly describe each process in turn, but don't construe any one as being independent or more important than the others. Furthermore, it appears that these processes occur concurrently, rather than strictly ordered as in a traditional life-cycle model or partially ordered as in a spiral process model.

Requirements analysis and specification

Software requirements analysis helps identify the problems a software system should address and the form solutions might take. Requirements specification identifies an initial mapping of problems to system-based solutions. In FOSS development, how does requirements analysis occur, and where and how are requirements specifications described?

Studies to date have yet to find records of formal requirements elicitation, capture, analysis, and validation—the kind suggested by modern software engineering textbooks—in

any of the four communities.[4] In general, you can't find them on FOSS Web sites or in "requirements specification" documents. What studies have found and observed is different.

FOSS requirements take the form of threaded messages or discussions on Web sites that are available for open review, elaboration, refutation, or refinement. Requirements analysis and specification are implied activities. They routinely emerge as a by-product of community discourse about what its software should or shouldn't do and who'll take responsibility for contributing new or modified system functionality. The requirements appear as after-the-fact assertions in private and public email discussion threads, ad hoc software artifacts (such as source code fragments included in a message), and site content updates that continually emerge.[4,11] More conventionally, requirements analysis, specification, and validation aren't performed as a necessary task that produces a mandated requirements deliverable. Instead, you find widespread practices that imply reading and sense-making of online content. You find interlinked discourse "webs" that effectively trace, condense, and solidify into retrospective software requirements. All the while, the project is globally accessible to existing, new, or former FOSS project participants. Figure 1 shows an example of a *retrospective requirements specification*.

In short, requirements take these forms because FOSS developers implement their systems and then assert that certain features are necessary. They don't result from the explicitly stated needs of user representatives, focus groups, or product marketing strategists.

Coordinated version control, system build, and staged incremental release-review

Software version control tools such as the Concurrent Versions System—a FOSS system and document base[10]—are widely used in FOSS communities. Figure 2 shows one such FOSS repository on the Web.

Tools such as CVS serve as both a centralized mechanism for coordinating FOSS development and a venue for mediating control over which software enhancements, extensions, or upgrades will be checked in to the archive. If checked in, these updates will be available to the community as part of the alpha, beta, candidate, or official released versions, as well as the daily-build release.

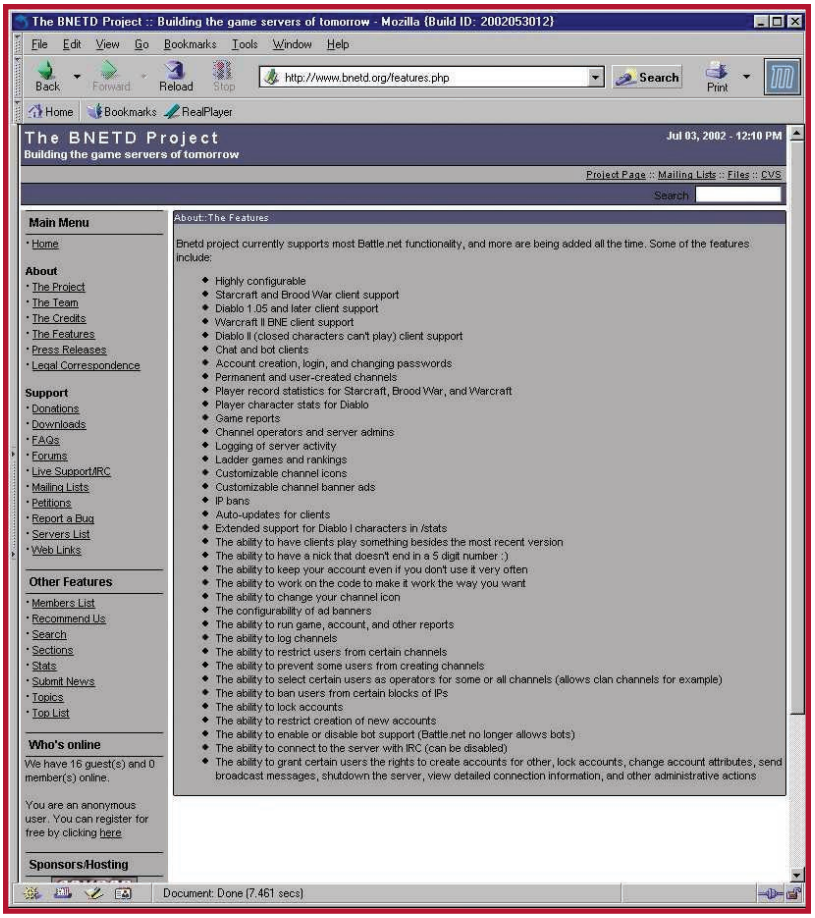


Figure 1. Computer-game software requirements specified as retrospectively asserted features. (figure courtesy of www.bnetd.org, July 2002)

Software version control, as part of a software configuration management activity, is recurrent. It requires coordination but lets you stabilize and synchronize dispersed, somewhat invisible development. This coordination is necessary because decentralized code contributors and reviewers might independently contribute software updates or reviews that overlap, conflict, or generate unwanted side effects.

Each project team or CVS repository administrator must decide what can be checked in and who can and can't check in new or modified software source code content. Some projects make these policies explicit through a voting scheme,[9] and in other projects they remain informal, implicit, and subject to negotiation with the designated module or version owner. In either case, the team must coordinate version updates for a new system build and release to take place. Subsequently, developers who want to submit updates to the community's shared repository rely primarily on online discussions in *lean media* form, such as threaded email messages posted on a site,[5] rather than having to deal with onerous system configuration control committees or seemingly arbitrary product delivery schedules. So, joint use of ver-

Figure 2. A view into a Web-accessible CVS (Concurrent Versions System) configuration archive of software source code files for the game Quake. (figure courtesy of the QuakeForge Project)

sioning, system building, online communication, and file-browsing and file-transfer tools mediates the process of coordinated version control, system build, release, and review.

Maintenance as evolutionary redevelopment, reinvention, and revitalization

Software maintenance—adding and subtracting system functionality, debugging, restructuring, tuning, conversion (for example, internationalization), and migration across platforms—is a widespread, recurring process in FOSS development communities. Perhaps this is no surprise, considering maintenance is generally viewed as the major activity associated with a software system across its life cycle. However, the traditional label of software maintenance doesn't quite fit what you see occurring in different FOSS communities. Instead, it might be better to characterize the overall evolutionary dynamic of FOSS as *reinvention*. Reinvention occurs through sharing, examining, modifying, and redistributing concepts and techniques that have appeared in closed-source systems, research and textbook publications, conferences, and developer-user discourse across multiple FOSS projects. Thus, reinvention is a continually emerging source of adaptation, learning, and improvement in FOSS functionality and quality.

FOSS systems seem to evolve through minor improvements or mutations that are expressed, recombined, and redistributed across many releases with short life cycles. FOSS end users who act as developers or maintainers continually produce these mutations. They appear initially in daily system builds. The modifications or updates are then expressed as tentative alpha, beta, or release versions that might survive redistribution and review. Then, they might be recombined and reexpressed with other mutations in producing a new, stable release version. As a result, these mutations articulate and adapt a FOSS system to what its user-developers want it to do while reinventing the system.

FOSS systems *coevolve* with their development communities; one's evolution depends on the other's. In other words, a project with few developers (most typically one) won't produce and sustain a viable system unless or until the team reaches a critical mass of between five and 15 core developers. If this happens, the system might be able to grow in size and complexity at a sustained exponential rate, defying the laws of software evolution that have held for decades.[12]

Closed-source software systems thought to be dead or beyond their useful product life or maintenance period may be *revitalized* through redistributing and opening their source code. However, this might only succeed in application domains with devoted, enthusiastic user-developers who are willing to invest time and skill to keep their cultural heritage alive. The Multiple Arcade Machine Emulator site (www.mame.net) for vintage arcade games shows that thousands of computer arcade games from the 1980s and 1990s are being revitalized through migration to FOSS-system support.

Project management and career development

FOSS development teams can take the organizational form of *interlinked layered meritocracies* operating as a dynamically organized but loosely coupled virtual enterprise.[13] A layered meritocracy[9] is a hierarchical organizational form that centralizes and concentrates certain kinds of authority, trust, and respect for experience and accomplishment within the team. However, it doesn't imply a

single authority, because decision-making can be shared among core developers who act as peers at the top echelon. Instead, meritocracies tend to embrace incremental innovations, such as evolutionary mutations to an existing software code base, over radical ones. Radical change involves exploring or adopting untried or sufficiently different system functionality, architecture, or development methods. A minority of code contributors who challenge the core developers' status quo might advocate radical changes. However, their success usually implies creating and maintaining a separate version of the system and potentially losing a critical mass of other FOSS developers. So incremental mutations tend to win out over time.

Figure 3 illustrates the form of a meritocracy common to many FOSS projects.[4] In this form, software development work appears to be logically centralized while physically distributed in an autonomous and decentralized manner.[13] However, it's neither simply a "cathedral" nor a "bazaar."[1] Instead, when layered meritocracy operates as a virtual enterprise, it relies on *virtual project management* to mobilize, coordinate, control, build, and assure the quality of FOSS development activities. It could invite or encourage system contributors to come forward and take a shared, individual responsibility that'll serve to benefit the FOSS collective of user-developers. VPM requires several people to act as team leader, subsystem manager, or system module owner in either a short- or long-term manner. People take roles on the basis of their skill, accomplishments, availability, and belief in community development. Figure 4 shows an example of VPM.

Project participants higher up in the meritocracy have greater perceived authority than those lower down. But these relationships are only effective if everyone agrees on their makeup and legitimacy. Administrative or coordination conflicts that can't be resolved can end up either splitting or forking a new system version. Then the conflicting participants must take responsibility for maintaining that version by reducing their stake in the ongoing project or by simply conceding the position in conflict.

VPM exists in FOSS communities to enable effective control via community decision-making and Web site and CVS repository administration. Similarly, it exists to mobilize and sustain the use of privately owned resources that the community can use (for example, Web servers, network access, site administrator labor, skill, and effort). Finally, some preliminary evidence suggests that, compared to projects with traditional software project management,

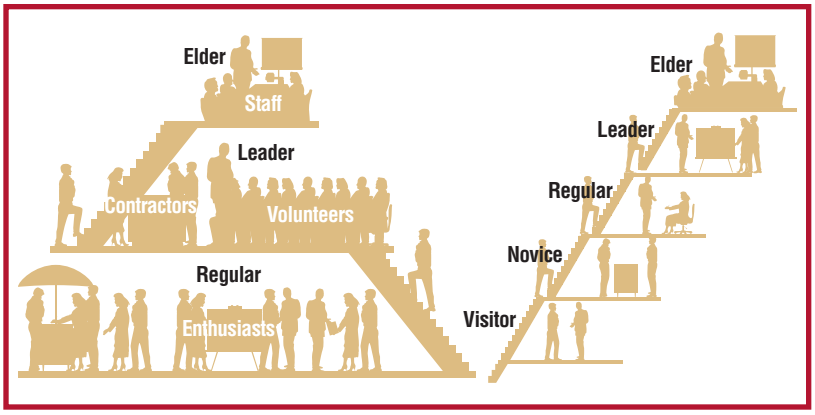


Figure 3. A layered meritocracy and role hierarchy.

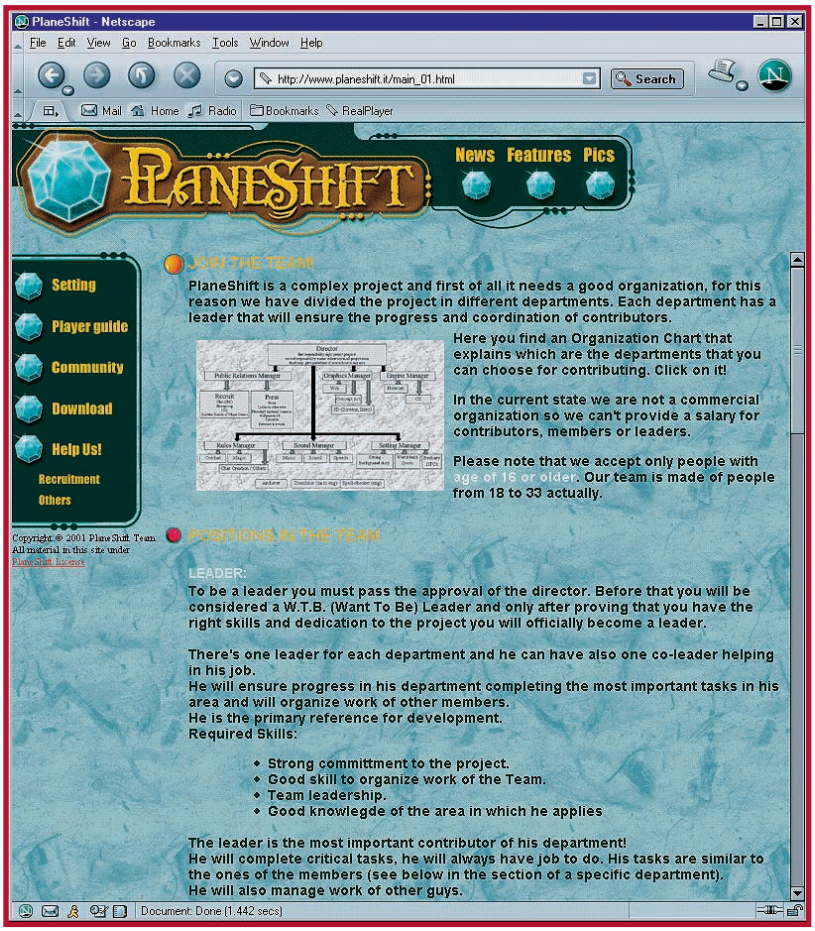


Figure 4. A description of how a FOSS computer game development project organizes and manages itself. This statement serves as an organizational surrogate to denote administrative authority, and includes an invitation to those who seek such project authority. (figure courtesy of PlaneShift)

Figure 5. This page highlights career development opportunities for would-be computer game developers via open source game mods. (figure courtesy of Epic Games)

FOSS projects can produce higher quality systems,[3] perhaps owing to VPM.

Traditional software project management stresses planning and control. Lawrence Lessig observes that source code intentionally or unintentionally achieves a mode of social control over those who use it.[15] So, in the case of FOSS development, Lessig's observation suggests that source code controls or constrains user–system interaction, while the code in software development tools, Web sites, and project assets controls, constrains, or facilitates developer interaction with the evolving FOSS system code. CVS enables some form of social control. However, the fact that these systems' source codes are freely available means that user-developers can examine, revise, and redistribute patterns of social control and interaction, thus favoring one form of project organization and user–system interaction over others. Thus, this dimension of VPM is open to manipulation by core developers. They can encourage certain patterns of development and social control and discourage ones that might not advance the collective needs of project participants.

FOSS developers have complex motives for being willing to allocate their time, skill, and effort to their systems' ongoing evolution. They might simply think the work is fun, personally rewarding, or a means to exercise and improve their technical competence in a way that they can't in their formal jobs or fields.[6] In FOSS computer game communities, "people even get hired for doing these things," as Figure 5 shows. Some FOSS developers create computer game modifications (*game mods*) that widely circulate and generate substantial sales revenue for the game's proprietary vendor, and they sometimes share in the profits.[8] Furthermore, being a central node in a network of software developers who interconnect multiple FOSS projects doesn't only bring social capital and recognition from peers. It also lets independent FOSS systems merge into larger ones that gain the critical mass of developers to grow even more and attract even larger user-developer communities. So, it might be surprising that more than 60 percent of the FOSS developers surveyed in a recent study[6] reported participating in two to 10 FOSS projects. This effectively interconnects not only independent system projects into a larger system architectures, but also interlinks their meritocracies, VPM practices, and social control. This enables the collective system and community to grow more robust together.

Software technology transfer and licensing

Software technology transfer is an important and often neglected process in the academic software engineering community. However, the diffusion, adoption, installation, and routine use of FOSS software systems and their Web-based assets are central to the systems' ongoing evolution. Transferring FOSS technology from existing Web sites to organizational practice is a community and project team-building process.[14] FOSS developers publicize and share their project assets by adopting and using FOSS project Web sites—a communitywide practice. You can build these Web sites using FOSS content management systems (such as PhP-Nuke) and serve them using FOSS Web servers (Apache), database systems (MySQL), or application servers (JBoss). User-developers are increasingly accessing these sites via FOSS Web browsers (Mozilla). Furthermore, ongoing FOSS proj-

ects might use dozens of FOSS development tools as stand-alone systems (CVS), integrated development environments (NetBeans or Eclipse), or their own application's subsystem components. These projects similarly employ asynchronous project communications systems that are persistent, searchable, traceable, public, and globally accessible.

FOSS technology transfer isn't an engineering process—at least not yet. It's instead a sociotechnical process that entails the development of constructive social relationships; informally negotiated social agreements; and a routine willingness to search, browse, download, and try out FOSS assets. It's also a commitment to continually participate in public, Web-based discourse and shared representations about FOSS systems, much like the other processes identified earlier. Community building and sustained participation are essential, recurring activities that let FOSS persist without centrally planned and managed corporate software development centers.

FOSS systems, development assets, tools, and project Web sites serve as a venue for socializing, building relationships and trust, sharing, and learning with others. Some open source software projects have made developing such social relationships their primary project goal. Figure 6 shows such a system, in which developers took an existing networked game system and created an open source game mod that transformed it into a venue for social activity. Many contemporary visual artists are also creating game mods as the basis for new art works (see examples at www.selectparks.net).

An overall, essential part of what enables the transfer and practice of FOSS development, and what distinguishes it from traditional software engineering, is the use and reiteration of FOSS public licenses. More than half of the 60,000 FOSS projects registered at SourceForge use the GNU General Public License. The GPL preserves and reiterates the beliefs and practices of sharing, examining, modifying, and redistributing FOSS systems and assets as property rights for collective freedom. Open source software projects that comingle assets that weren't created as free property have instead adopted variants that relax or strengthen the rights and conditions the GPL lays out. Visit www.opensource.org or www.creativecommons.org for general information on how to create these licenses.

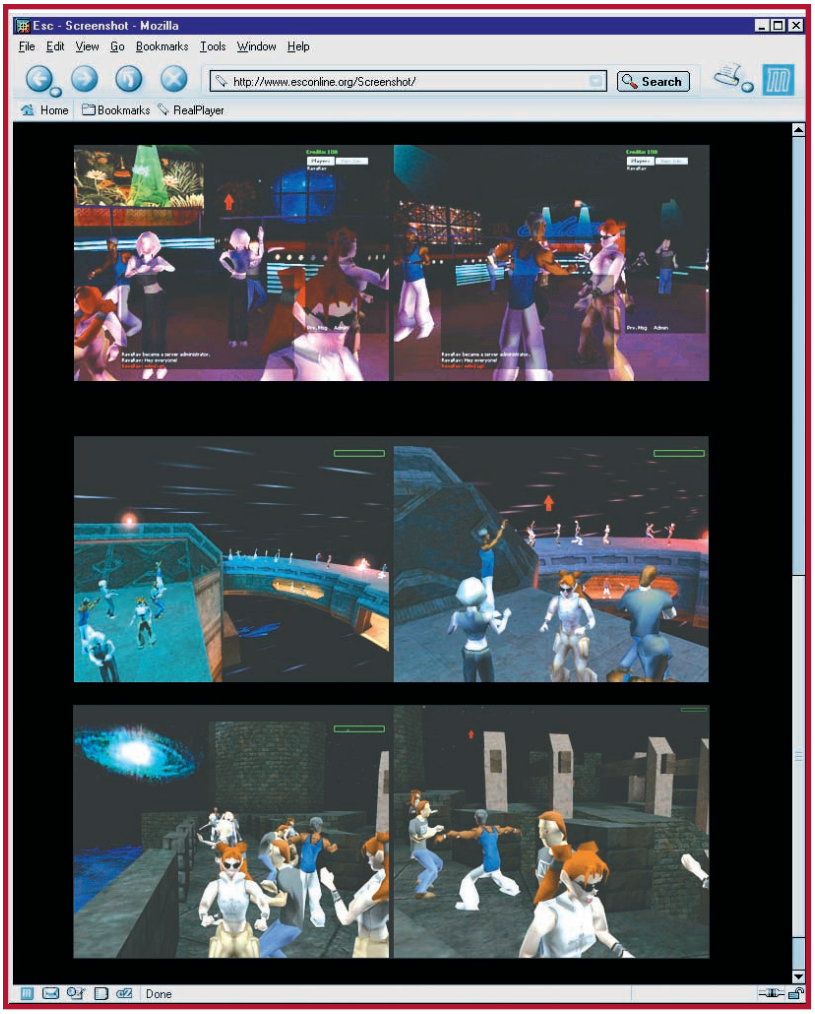


Figure 6. A first-person shooter game (Unreal Tournament) that's been modified and transformed into a 3D virtual environment for socializing and virtual dancing with in-game avatars. (figure courtesy of Martin C. Martin)

Free and open source software development practices give rise to a new view of how complex software systems can be constructed, deployed, and evolved. FOSS projects don't adhere to traditional software engineering life-cycle principles from modern textbooks. They rely on lean electronic communication media, virtual project management, and version management mechanisms to coordinate globally dispersed development efforts. They coevolve with their development communities, which reinvent and transfer software technologies as part of their team-building process. Practices to propagate FOSS technology and culture are intertwined and mutually situated to benefit motivated participants and contributors.

So, software engineering managers and developers working in traditional proprietary, closed-source, centrally managed, and colocated software development centers might recognize that viable alternatives exist to the practices and principles they've been following. These FOSS processes offer new directions for developing complex software systems.

About the Author

Walt Scacchi is a senior research computer scientist and research faculty member at the Institute for Software Research and director of research for the Laboratory for Game Culture and Technology, both at the University of California, Irvine. His research interests include open source software development, software process engineering, computer game culture and technology, and organizational studies of system development. He received his PhD in information and computer science from UC Irvine. He is a member of the AAAI, ACM, and IEEE. Contact him at the Institute for Software Research, Univ. of California, Irvine, Irvine, CA 92697-3425; wscacchi@ics.uci.edu; www.isr.uci.edu/open-source-research.html.

Acknowledgments

Grants IIS-0083075, ITR-0205679, ITR-0205724, and ITR-0350754 from the US National Science Foundation supported this research. Andrew Henderson and James Neighbors commented on an earlier draft.

References

1. C. DiBona, S. Ockman, and M. Stone, *Open Sources: Voices from the Open Source Revolution*, O'Reilly and Associates, 1999.
2. S. Williams, *Free as in Freedom: Richard Stallman's Crusade for Free Software*, O'Reilly and Associates, 2002.
3. A. Mockus, R.T. Fielding, and J. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Trans. Software Eng. and Methodology*, vol. 11, no. 3, July 2002, pp. 309–346.
4. W. Scacchi, "Understanding the Requirements for Developing Open Source Software Systems," *IEE Proc.—Software*, vol. 149, no. 1, Feb. 2002, pp. 24–39.
5. Y. Yamauchi et al., "Collaboration with Lean Media: How Open-Source Software Succeeds," *Proc. Computer Supported Cooperative Work Conf.* (CSCW 00), ACM Press, pp. 329–338.
6. A. Hars and S. Ou, "Working for Free? Motivations for Participating in Open-Source Software Projects," *Int'l J. Electronic Commerce*, vol. 6, no. 3, Spring 2002, pp. 25–39.
7. S. Viller and I. Sommerville, "Ethnographically Informed Analysis for Software Engineers," *Int'l. J. Human–Computer Studies*, vol. 53, no. 1, July 2000, pp. 169–196.
8. C. Cleveland, "The Past, Present, and Future of PC Mod Development," *Game Developer*, vol. 8, no. 2, Feb. 2001, pp. 46–49.
9. R.T. Fielding, "Shared Leadership in the Apache Project," *Comm. ACM*, vol. 42, no. 4, Apr. 1999, pp. 42–43.
10. K. Fogel, *Open Source Development with CVS*, Coriolis Press, 1999.
11. D. Truex, R. Baskerville, and H. Klein, "Growing Systems in an Emergent Organization," *Comm. ACM*, vol. 42, no. 8, Aug. 1999, pp. 117–123.
12. M.M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," *Proc. IEEE*, vol. 68, no. 9, Sept. 1980, pp. 1060–1078.
13. J. Noll and W. Scacchi, "Supporting Software Development in Virtual Enterprises," *J. Digital Information*, vol. 1, no. 4, Jan. 1999, http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll.
14. A.J. Kim, *Community-Building on the Web: Secret Strategies for Successful Online Communities*, Peachpit Press, 2000.
15. L. Lessig, *CODE and Other Laws of Cyberspace*, Basic Books, 1999.

OpenEC/B: Electronic Commerce and Free/Open Source Software Development

Walt Scacchi
Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697-3425 USA
http://www.ics.uci.edu/~wscacchi

Abstract

This report investigates Open Source E-Commerce or E-Business capabilities. This entails a case study within one firm that has undertaken an organizational initiative to develop, deploy, use, and support free/open source software systems for Enterprise Resource Planning (ERP), E-Commerce (EC) or E-Business (EB) services. The objective is to identify and characterize the resource-based software product development capabilities that lie at the center of the initiative.

Introduction

This paper presents and analyzes a case study that examines how a firm can support an E-Commerce or E-Business initiative that builds from free/open source software (FOSS) product development capabilities. Such capabilities may focus, for example, on back office activities associated with corporate financial operations, or on front office activities associated with customer relationship management. Alternatively, the focus may be directed as an organizational system where wireless, mobile, or p2p capabilities are sought.

The study employs a resource-based view of the organizational system involved in developing an open source EC/EB software products or application systems. The analysis and results of the case study focus attention to data that characterizes the organization's resource-based product development capabilities. This case study examines the GNUenterprise.org project. This study serves as a point of departure to explicate the concept of *Open EC/B* introduced in this paper. Open EC/B results from combining OSSD concepts, techniques, and tools with those for EC and EB.

Case Study: *GNUenterprise.org* and the development of FOSS ERP software

GNUenterprise.org is an international virtual organization for software development [Crowston and Scozzi 2002, Noll and Scacchi 1999] based in the U.S. and Europe that is developing a free, open source Enterprise Resource Planning (ERP) systems and related E-Business capabilities.

As such, these conditions make this study unique in comparison to previous case studies of EC or EB initiatives, which generally assume the presence of a centralized administrative authority and locus of resource control common in most large firms. Nonetheless, we still need a better understanding of what resource-based capabilities are brought to bear on the development and deployment of EB and ERP software by GNUenterprise.org. Subsequently, what follows is a description of key resources being employed throughout GNUenterprise.org to develop and support the evolution of the GNUe software modules.

The following sections present an interpretive analysis of the case study, as is appropriate for the kinds of data and descriptions that have been presented and in related studies [cf. Scacchi 2001, 2002, Skok and Legge 2002]. One category of challenges to Open EC/B that is apparent is that denoting resource-based capabilities.

Resources and Capabilities for Open EC/B

What kinds of resources or business capabilities are needed to help make Open EC/B efforts more likely to succeed? Based on what was observed in the GNUenterprise.org case study, the following kinds of resources enable the development of both FOSS ERP/EB software and community that is sustaining its evolution, application and refinement:

Personal software development tools and networking support

FOSS developers, end-users, and other volunteers provide their own personal computing resources in order to access or participate in a FOSSD community 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 the community. The sustained commitment of personal resources helps subsidize the emergence and evolution of the community, and its shared (public) information resources. It also helps create recognizable

shares of the commons that are linked (via hardware, software, and Web) to community infrastructure.

Beliefs supporting FOSSD

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 share, learn, modify, and redistribute the evolving results from a FOSSD project. Without such belief, it seems unlikely that there could be "free" and "open source" software development projects [DiBona, Ockman and Stone, 1999, Williams 2002]. However, one important consideration that follows is what are the consequences from such belief, and how are these consequences are put into action.

In looking across the case study data, many kinds of actions or choices emerge from the development of FOSS. Primary among them is *freedom of expression and choice*. Neither of these freedoms is explicitly declared, assured, or protected by free software copyright or community intellectual property rights. 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 (personal tool choice 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 community ownership responsibility), and expressing what can be said to whom with or without reservation (modulated by trust and accountability). Shared belief and practice in freedom of expression and choice are part of the organizational culture that characterizes a community project like GNUenterprise.org [Elliott and Scacchi 2004]. Subsequently, putting these beliefs and cultural resources into action builds both community and FOSS.

Competently skilled and self-organizing FOSS developers

Developing complex software modules for ERP applications requires skill and expertise in the domain of EB and EC. Developing these modules 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 bug reports also contributes to the development of such a system architecture. These are among the valuable professional skills that are mobilized, brought to, or drawn to FOSSD community projects like GNUenterprise.org [cf. Crowston and Scozzi 2002]. 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. Within in-house development projects, software application developers and end-users often are juxtaposed in opposition to one another. Danziger [1979] referred to this concentration of software 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 FOSSD community like GNUenterprise.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. Instead, FOSSD participants organize around the expertise, reputation, and accomplishments of core developers, secondary contributors, and tertiary reviewers and other volunteers.

Participants nearer the core have greater control and discretionary decision-making authority, compared to those further from the core. 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 community 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 free open source ERP and EB software modules, so long as they are active participants in their community project.

Discretionary time and effort of developers

Are OSS developers working for "free" or for advancing their career and professional development? Following the survey results of Hars and Ou [2002] and others

[Lerner and Tirole 2000, Hann, et al. 2002], 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. What we have found in GNUenterprise.org appears consistent with their observations. These include include not only self-determination, peer recognition, community identification, and self-promotion, but also belief in inherent value of free software [cf. DiBona, Ockman, and Stone, 1999, 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. OSS 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 community, particularly with regard to software modules for which they have declared responsibility to maintain or manage as a core developer.

In the practice of peer recognition, a developer becomes recognized as an increasingly valued community contributor as a growing number of their contributions make their way into the core software modules [Bergquist and Ljundberg 2001]. In addition, nearly two-thirds of OSS developers work on 1-10 *additional* OSSD projects [Hars and Ou 2002], which also reflects a growing social network of alliances across multiple free, OSSD projects [cf. Monge, *et al.* 1998]. The project contributors who span multiple project communities can serve as "social gateways" that increase the community's mass [Marwell and Oliver 1993] and opportunity for inter-project software composition and bricolage. It also enables and empowers their recognition across multiple communities of FOSSD peers.

In building community identification, project participants build shared domain expertise, and identify who is expert in knowing how to do what [cf. Ackerman and Halverson 2000]. Interlinked contents and persistent communicated messages help point to who the experts and core contributors are.

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 towards the center or core of the development effort requires not only the time and effort of a contributor, but also the ability to convince others as to the significance of the contributions. 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 observed in GNUenterprise.org is found in the freedoms and beliefs in FOSSD that are shared, reiterated and put into observable interactions. If a community participant fails to sustain or reiterate the freedoms and beliefs institutionalized in the GPL, then it is likely the person will leave the project and community. But understanding how these freedoms and beliefs are put into action points to another class of (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.

Trust and social accountability mechanisms

Developing complex software modules for ERP, EB, or EC applications requires trust and accountability among project participants. Though trust and accountability in a FOSSD project may be invisible resources, ongoing software and community development work occur only when these intangible resources and mechanisms for social control are present [cf. Hertzum 2002].

The intangible resources arise in many forms. They include assuming ownership or responsibility of a community software module, voting on the approval of individual action or contribution to community software [Fielding 1999], shared peer reviewing [DiBona, Ockman and Stone 1999], and by contributing gifts [Bergquist and Ljundberg 2001] that are reusable and modifiable public goods [Olson 1971]. They also exist through the community'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 community's mass becomes sufficient to insure that individual trust and accountability to the project community 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 project management regime that would schedule, budget, staff, and control the project's development trajectory with varying degrees of administrative authority and technical competence.

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 resources that are comparatively easy to use, and immediately familiar to those who want to join the community project. However, the contents they embody require extensive review and comprehension by a developer before core contributions can be made. The most common informalisms include community communications and messages within Email, threaded Email discussion forum, news postings, community digests, and instant messaging chat. They also include scenarios of usage as linked Web pages, how-to guides, to-do lists, FAQs, and other itemized lists, as well as traditional system documentation and external publications. FOSS community property licenses also help to define what software or related project content are protected resources that can subsequently be shared, examined, modified, and redistributed. Finally, open software architectural designs, scripting languages like Perl and PhP, and the ability to either plug-in or 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 project related Web sites or portals. These Web environments are also software informalisms [Scacchi 2002]. A project's Web presence helps make visible the community's information infrastructure and the array of information resources that populate it. These include OSSD community project Web sites (e.g., SourgeForge.net, Savanah.org, and Freshment.org), community software Web sites (PhP-Nuke.org), and project Web site (www.GNUenterprise.org), as well as embedded project source code Webs (directories), project repositories, and software bug reports and issue tracking data base.

Together, these software informalisms constitute a substantial collection of information resources and artifacts that are produced, used, consumed, or reused within and across FOSSD projects.

FOSSD capability enabling free, open ERP and EB systems

The array of social, technological, and informational resources that enable a FOSSD project is substantial. However, they differ in kind and form from the traditional enterprise resources that are provided to support proprietary, closed source software systems. These traditional resources are money (budget), time (schedule), skilled development staff, project managers (administrative authority), quality assurance (QA) and testing groups, documentation writers, computer hardware and network maintainers, and others. FOSSD projects seem to get by with comparatively small amounts of money, though subsidies of various kinds and sources are present and necessary. They also get by without explicit schedules, though larger projects may announce target release dates, as well as (partially) order which system functions or features will be included in some upcoming versions, for some target release. Further, they get by without the rule-making and decision-making authority of project managers, who may or may not be adept at empowering, coaching, or rewarding development staff to achieve corporate software development goals. The remaining resources are provided within a FOSSD effort via subsidies, sponsorship, or volunteer effort.

Thus, the resources for FOSSD efforts are different: they are not mobilized, allocated, or otherwise brought to bear in the manner traditional to the development of proprietary, closed source software systems. Hopefully, it should be clear that the differences being highlighted are not based simply on a comparison of functionality or features visible in the development or use of open vs. close source software products. As such, the resource-based capability for developing FOSS components or modules for ERP, EB and EC applications is different.

Conclusions

Two main conclusions can be drawn from the study, data, and analysis presented in this report.

First, this study identified and introduced a new concept called *OpenEC/B*. OpenEC/B denotes the integration of FOSSD resources, products, and processes, with the existing or emerging capabilities for Electronic Commerce/Business. This concept and its consequences are explained in the case study and analysis. No prior case studies of EC/EB have identified or addressed whether or how OSS methods might be applied or integrated with EC/EB, at least beyond the use of OSS Web servers or Web-site content management systems. Thus, there is an opportunity for firms to begin considering whether these results merit timely consideration or exploratory investments. For example, companies offering consumer products or high value, information technology based products and services may begin to consider whether OpenEC/B capabilities that offer lower purchase prices, lower total cost of ownership, and higher quality represent new market entry or new product differentiation opportunities. Similarly, companies may find FOSSD represents a highly innovative approach to software product development that marries the best capabilities from both

private investment and collective action [Marwell and Oliver 1993, Olson 1971, von Hippel and von Krogh 2003]

Last, this study identifies resources and resource-based capability for OpenEC/B that may explain or predict (a) what's involved, (b) how it works, or (c) what conditions may shape the longer-term success or failure of such efforts. In simple terms, these resources include time, skill, effort, belief, personal and corporate subsidies, and community building on the part of those contributing as developers and users of OpenEC/B systems and techniques. Of these, *belief* in the freedoms that open source system development allows appears central. Developers and users who believe in the promise and potential of OpenEC/B systems are willing to allocate (or volunteer) their time and apply their skills to make the effort of developing or using open source systems a viable and successful course of action. Thus companies seeking to invest in or exploit OpenEC/B techniques or systems must account for how it can most effectively cultivate an OpenEC/B culture, belief system, and community of practice, as part of their strategic choice.

Acknowledgements*:* The research described in this report is supported by grants from the NSF Industry/University Research Cooperative CRITO Consortium, National Science Foundation # 0083075, #0205679, #0205724, and #0350754 and from the Defense Acquisition University by contract N487650-27803. No endorsement implied.

References

M. Ackerman and C. Halverson, Reexamining Organizational Memory, *Communications ACM*, 43(1), 59-64, January 2000.

M. Bergquist and J. Ljungberg, The Power of Gifts: Organizing Social Relationships in Open Source Communities, *Info. Systems J.*, 11(4), 305-320, 2001.

K. Crowston and B. Scozzi, Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings--Software*, 149(2), 3-17, 2002.

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

C. DiBona, S. Ockman and M. Stone, *Open Sources: Voices from the Open Source Revolution*, O'Reilly Press, 1999.

M. Elliott and W. Scacchi, Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, 152-172, Idea Publishing, Pittsburgh, PA, 2004.

R. Fielding, Shared Leadership in the Apache Project, *Communications ACM*, 42(4), 42-43, 1999.

I-H. Hann, J. Roberts, S. Slaughter, and R. Fielding, Why Do Developers Contribute to Open Source Projects? First Evidence of Economic Incentives, *Proc. 2nd Workshop on Open Source Software Engineering*, Orlando, FL, May 2002.

A. Hars and S. Ou, Working for Free? Motivations for Participating in Open-Source Projects, *Intern. J. Electronic Commerce*, 6(3), 25-39, 2002.

M. Hertzum, The importance of trust in software engineers' assessment and choice of information sources, *Information and Organization*, 12(1), 1-18, 2002.

J.Lerner and J. Tirole, Some Simple Economics of Open Source, *Journal of Industrial Economics*, 52, 2002.

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

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

J. Noll and W. Scacchi, Supporting Software Development in Virtual Enterprises, *Journal of Digital Information*, 1(4), February 1999.

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

R. Pavlicek, *Embracing Insanity: Open Source Software Development*, SAMS Publishing, Indianapolis, IN, 2000.

W. Scacchi, Redesigning Contracted Service Procurement for Internet-based Electronic Commerce: A Case Study, *J. Information Technology and Management*, 2(3), 313-334, 2001.

W. Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(2), 24-39, 2002.

W. Skok and M. Legge, Evaluating Enterprise Resource Planning (ERP) System using an Interpretive Approach, *Knowledge and Process Management*, 9(2), 72-82, 2002.

E. von Hippel and G. von Krogh, Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science, *Organization Science*, 14(2), 209-223, 2003.

S. Williams, *Free as in Freedom: Richard Stallman's Crusade for Free Software*, O'Reilly Books, Sebastopol, CA, 2002.

Understanding Open Source Software Evolution

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA USA 92697-3425
wscacchi@uci.edu
July 2004
Previous versions: April 2003, August 2003, January 2004, June 2004

1. Introduction

This chapter examines the evolution of open source software and how their evolutionary patterns compare to prior studies of software evolution of proprietary (or closed source) software. Free or open source software (F/OSS) development focuses attention to systems like the GNU/Linux operating system, Apache Web server, and Mozilla Web browser, though there are now thousands of F/OSS projects underway. As these systems are being ever more widely used, questions regarding their evolution are of considerable interest.

This chapter is organized around four themes. First, it presents a brief survey of empirical studies of software evolution. As the majority of published studies of this kind are associated with the development of the laws of software evolution due to Lehman and colleagues, the kinds of findings they provide are described. Additionally, a sample of other empirical studies of software evolution are provided as well, in order to round out what is presently known about software evolution, at least in terms of studies of closed source software systems developed within centralized software development centers.

Second, it presents selected data and evidence that has begun to appear that characterizes change and evolution patterns associated with the evolution of F/OSS. Along the way, attention shifts to an analysis of where, how, and why the evolution of F/OSS does or does not conform to prior empirical studies, models, or theories of software evolution. Without revealing too much at this point, it is fair to say that there are patterns of data from studies of F/OSS that are not fully explained by prior studies of software evolution, as presently stated.

Third, it presents a brief review of models and theories of evolution from domains outside of software. This will help facilitate understanding of some of the challenges and alternative historical groundings that might be used to shape our collective understanding of how to think more broadly about software evolution, as well as the significance of theorizing about it.

The fourth and last section addresses whether it is necessary to reconsider the models, laws, and theory and how they can be modified and supplemented to better account for the observations and findings emerging in studies of new software development processes and environments, such as those associated with the development of F/OSS. Prior models

of software evolution were developed based on careful study of conventional, closed source software system that evolve within industrial settings. Studies of F/OSS examine systems that typically are evolved outside of industrial settings, though some F/OSS systems are used in industrial settings, even though that are evolved in non-industrial environments. However, it is appropriate to consider how to update and revise the models, laws, and theory of software evolution to better account for both open and closed-source software system are being evolved inside or outside of industrial settings.

As such, the remainder of this chapter progresses through each of these themes in the order presented here.

2. Empirical Studies of Software Evolution

To understand the state of the art in the development of a theory of software evolution, and whether and how it might be extended, it is necessary to identify and describe what empirical studies of software evolution have been reported.

2.1 Studies of the Laws of Software Evolution

The most prominent studies of software evolution have been directed by M.M. Lehman and colleagues over a 30 year period dating back to the mid-1970's. The studies have given rise to eight laws of software evolution, as formulated and refined by Lehman and colleagues [Lehman, *et al*., 1980-2001]. These laws are the result of careful and challenging empirical studies of the evolution of large-scale software systems found in a variety of corporate-based settings. These laws seek to consistently account for observed phenomena regarding the evolution of software releases, systems, and E-Type applications, as defined by Lehman and colleagues. The laws and theory can be formulated in a manner suitable for independent test and validation, or refutation [Lakatos 1976, Popper 1961], but this requires making assumptions about details that are not explicitly stated in the laws. Thus there are many challenges in how such empirical testing of these laws should be performed (e.g., how many or what kinds of software systems constitute an adequate or theoretically motivated sample space for comparative study), what the consequences for refutation may be (rejection or reformulation of the laws/theory), and whether or how the laws and theory might be refined and improved if new or contradictory phenomena appear [cf. Glaser and Strauss 1976, Yin 1994].

The published studies by Lehman and colleagues provide data from evolution of releases primarily from five software systems: two operating systems (IBM OS 360, ICL VME Kernel), one financial system (Logica FW), two versions of a large real-time telecommunications system, and one defense system (Matra BAE Dynamics). Other studies have also been conducted and found to yield consistent growth models, but their results are not widely available. The data is summarized as a set of growth curves, as described in Perry and Ramil [2004]. In plotting these growth curves as graphs, the X-axis denotes the sequence number of the software release that was analyzed, while the Y-axis denotes the growth of the size of the system (e.g., measured in the number of modules) after the first release. The graphs suggest that during its evolution (or maintenance process), a system tracks a growth curve that can be approximated either as linear or inverse-square model [Turski 1996]. Thus, these data/curves explicate

conformity to the first six laws, in that they suggest continual adaptation via incremental growth, system complexity controls the growth rate in a constant/bounded (linear or inverse-square) manner. The last two laws addressing quality and feedback systems cannot be directly observed within the data, but may conform to observations made by Lehman and colleagues about these systems. Therefore, this data set and the diversity of data substantiates and supports the laws.

However, it is unclear whether such a data set is a representative sample of different kinds/types of software systems, or whether the laws can be interpreted as providing theoretical guidance for what kinds/types of software systems to study. It may be apparent that the majority of systems are large or very large software systems developed and maintained in large corporate settings, that the customers for such systems are also likely to be large enterprises (i.e., they are not intended as software for a personal or hand-held computer). In addition, some of the software systems or associated data that were examined in the studies by Lehman and colleagues are confidential, and thus are not open for public inspection, or independent examination and assessment. Subsequently, students and other scholars cannot readily access these systems or data for further study.[1]

2.2 Other Empirical Studies of Software Evolution

Many other empirical studies have been conducted and published. Here attention is directed to a sample of these studies where non-open source software systems were being investigated. This is mainly intended to see if other studies of software evolution conform to, refute, or otherwise extend and refine the laws and theory of software evolution..

Bendifallah and Scacchi [1987] present qualitative data and analysis of two comparative case studies revealing that similar kinds of software systems in similar kinds of organizational settings have different evolutionary trajectories. They report the differences can be explained by how system maintainers and end-users deal with local contingencies in their workplace and career opportunities in the course of maintaining their software systems.

Tamai and Torimitsu [1992] present data and observations from a survey study of mainframe software system applications across product generations. Among other things, they report that software lifetime in their survey is on average about 10 years, the variance in application lifetime is 6.2, and that small software applications tend to have a shorter live on average. They also report that applications that constitute what they call *administration systems* (e.g, back office applications) live longer than *business supporting* (i.e., mission-critical) systems, and that application systems that replace previous generation systems grow by more than a factor of 2 compared to their predecessors. Last, they report that some companies follow policies that set the predicted lifetime of an application system at the time of initial release, and use this information in scheduling migration to next generation systems.

[1] Early data from Lehman's studies can be found in one of his books [Lehman and Belady 1985]. Unfortunately, the *unavailability* of empirical data from software measurements studies is in general all too common of an occurrence. However, studies of F/OSS may indicate a different future lies ahead regarding public data availability [cf. Koch and Schneider 2000, Robles-Martinez, Gonzalez-Barahona, *et al*., 2003].

Cusumano and Yoffie [1999] present results from case studies at Microsoft and Netscape indicating strong reliance on incremental release of alpha and beta versions to customers as business strategy for improving evolution of system features that meet evolving user requirements. They show that user satisfaction can improve and be driven by shortening the time interval between releases. They also find that unstable releases (e.g., alpha and beta versions) will be released to users as a way to enable them to participate in the decentralized testing and remote quality assurance, and thus affecting software evolution. Their study does not confirm or refute the laws of software evolution, but they introduce a new dynamic into software evolution by making the release activity an independent output variable rather than an input variable.

Gall, Jayazeri, *et al*., [1997] provide data and observations based on software product release histories from a study of a large telecommunications switching system. The growth of this system over twenty releases conforms to the general trends found in the data of Lehman and colleagues. However, they report that though global system evolution follows the trend and thus conforms to the laws, individual subsystems and modules do not. Instead, they sometimes exhibit significant upward or downward fluctuation in their size across almost all releases. Eick, Graves, *et al.*, [2001] also provide data demonstrating that source code decays unless effort and resources are allocated to prevent and maintain the system throughout the later stages of its deployment, and that the decay can be observed to rise and fall in different subsystems and modules across releases.

Kemerer and Slaughter [1999] provide a systematic set of data, analyses, and comparison with prior studies revealing that problems in software maintenance can be attributed to a lack of knowledge of the maintenance process, and of the cause and effect relationships between software maintenance practices and outcomes. However, they do observe that their data may be associated with the growth of system complexity and other outcomes over time, which they attribute to the laws observed by Lehman and colleagues.

Perry, Siy, and Votta [2001] report findings from an observational case study of the development of large telecommunications systems that indicates extensive parallel changes being made between software system releases. This notion of parallel changes that may interact and thus confound software maintenance activities is not accounted for in an explicit way by the laws of software evolution. Thus, it does introduce yet another organizational factor that may affect software evolution.

With the exception of Cusumano and Yoffie [1999], these studies either conform to or suggest extensions to the laws and theory of software evolution. Thus these conditions may point to the need for either revisions to the laws, or alternative theories of software evolution that may or may not depend on such laws.

3. Evolutionary Patterns in Open Source Software

F/OSS development has appeared and disseminated throughout the world of software technology, mostly in the last ten years. This coincides with the spread, adoption, and routine use of the Internet and World Wide Web as a global technical system. This infrastructure supports widespread access to previously remote information and software

assets, as well as the ability for decentralized communities of like-minded people to find and communicate with one another. This is a world that differs in many ways from that traditional to software engineering, where it is common to assume centralized software development locales, development work, and administrative authority that controls and manages the resources and schedules for software development and maintenance. Thus to better understand whether or how patterns of software evolution in the technical and social regime of F/OSS conform to or differ from prior studies or models of software evolution, then it is appropriate to start with an identification of the types of entities for F/OSS evolution, then follow with an examination of empirical studies, data and analyses of F/OSS evolution patterns.

3.1 Types of entities for studying F/OSS evolution

The scheme of objects types that are suitable to address in studies of software evolution have been identified in the studies by Lehman and colleagues over the years [cf. Lehman 1980, 2002]. The primary types of entities are software releases, systems, applications, development processes, and process models. Accordingly, each of these can be cast in terms of F/OSS as follows.

F/OSS Releases — Large F/OSS systems continue to grow over time and across releases. This suggests consistency with the sixth law of software evolution. Both stable and unstable F/OSS release product versions are being globally distributed in practice. Periodic alpha, beta, candidate, and stable releases are made available to users at their discretion, as are unstable nightly F/OSS build versions released for developers actively contributing software updates to a given release. F/OSS releases for multiple platforms are generally synchronized and distributed at the same time, though may vary when new platforms are added (in parallel). F/OSS releases thus evolve within a non-traditional process cycle between full stable releases. F/OSS releases are also named with hierarchical release numbering schemes, sometimes with three or four levels of nested numbering to connote stable versus unstable releases to different audiences. However, the vast majority of F/OSS systems, primarily those for small and medium size F/OSS systems, do not continue to grow or thrive, perhaps because the software is not intensively or widely used [[Capiluppi, Lago, and Morisio 2003].

F/OSS Systems – F/OSS systems or programs evolve from first statement of an application concept or a change required, to an existing system released and installed as an operational program text with its documentation. F/OSS systems may be small (<5K SLOC[2]), medium (5K-100K SLOC), large (100K-1000K SLOC) or very large systems (>1MSLOC), with large and very large systems being the fewest in number, but the most widely known. Most large or very large F/OSS systems or programs may exist in related but distinct versions/releases intended for different application platforms (e.g., MS Windows, Solaris, GNU/Linux, Mac OS X). Many F/OSS are structured as distributed systems, systems configured using scripts (e.g., using Perl, Python, Tcl), middleware, or as modules that plug-in to hosts/servers (e.g., Apache and Mozilla both support independently developed plug-in modules). Additionally, some F/OSS are dynamically

[2] Source Lines of Code, where 50 SLOC represents the equivalent of one printed page of source code, single spaced.

linked systems configured at run-time, when developed in a programming language like Java or others enabling remote service/method invocation.

F/OSS Applications – A much greater diversity and population of F/OSS applications are being investigated for evolution patterns. Those examined in-depth so far include the Linux Kernel, Debian Linux distributions[3], Mono, Apache Web server, Mozilla Web browser, Berkeley DB, GNOME user interface desktop, PostgreSQL DBMS, and about a dozen others[4]. Studies of F/OSS application populations, taxonomy, and population demographics for hundreds to upwards of 40K F/OSS systems have appeared [Madey, Freeh, and Tynan 2002].

F/OSS Process – F/OSS are developed, deployed, and maintained according to some software process. It is however unclear whether F/OSS processes, as portrayed in popular literature [DiBona, Ockman and Stone 1999], are intended only to be viewed as a monolithic process, just the top-level of a decomposable process, or whether specific software engineering activities have distinct processes which may also evolve, either independently or jointly. Furthermore, a small number of recent studies have begun to observe, describe and compare F/OSS development processes with those traditional to software engineering [Reis and Fortes 2002, Mockus, Fielding, and Herbsleb 2002, Scacchi 2002a,b, 2004] that point to differences in the activities and organization of the F/OSS process. In addition, F/OSS activities surrounding software releases may have their own distinct process [Erenkrantz 2003, Jensen and Scacchi 2003] that may not reflect the activities involved in the release of closed-source systems examined in the preceding section.

Models of F/OSS Process – Existing models of software development processes [Scacchi 2002b] do not explicitly account for F/OSS development activities or work practices [cf. Scacchi 2002a, 2002c, Jensen and Scacchi 2003]. Thus it is unclear whether models of software evolution processes that characterize closed-source software systems developed within a centralized administrative authority can account the decentralized, community-oriented evolution of F/OSS.

Overall, evolving software systems may be packaged and released in either open source or closed source forms. The packaging and release processes and technical system infrastructure may at times differ or be the same, depending on the software system application and development host (e.g., a Web site for open source, a corporate portal for closed source). But the decentralized community-oriented technological regime and infrastructure of F/OSS appears different than the world of the centralized corporate-centered regime and infrastructure of the closed source systems that have been examined

[3] A GNU/Linux distribution includes not only the Kernel, but also hundreds/thousands of utilities and end-user applications. Distributions are typically the unit of installation when one acquires GNU/Linux, while the Linux Kernel is considered the core of the distribution. However, many F/OSS applications are developed for non-Linux Kernel operating systems (e.g., MS Windows), thus assuming little/no coupling to the Linux Kernel.

[4] Smith, Capiluppi, and Ramil [2004] have published preliminary results from an ongoing comparative study of 26 OSS systems and applications. Such studies begin to suggest that future studies of software evolution will focus attention to F/OSS for a variety of reasons.

as the basis of the laws of software evolution. Nonetheless, the laws of software evolution seem to apply, at least at a very high level, in accounting for the evolution of F/OSS.

3.2 Patterns in Open Source Software Evolution Studies

Attention is now directed to examples of studies where F/OSS systems are being investigated, with the focus on how their results can be compared with those of Lehman and colleagues.

Godfrey and Tu [2000] provide data on the size and growth of the Linux Kernel (2M+ SLOC) from 1994-1999, and find the growth rate to be super-linear (i.e., greater than linear), as portrayed in Figures 1 through 3. They also find similar patterns in F/OSS for the Vim text editor. Schach, Jin, *et al*., [2002] report on the result of an in-depth study of the evolution of the Linux Kernel across 96 releases [cf. Godfrey and Tu 2000] indicating that module coupling (or interconnection) has been growing at an exponential (super-linear) rate. Their data are displayed in Figure 4. They predict that unless effort to alter this situation is undertaken, the Linux Kernel will become unmaintainable over time. Koch and Schneider [2000] studied the GNOME user interface desktop (2M+ SLOC) and provided data that shows growth in the size of the source code base across releases increases in a super-linear manner as the number of software developers contributing code to the GNOME code base grows. Data from their study is plotted in Figure 5.

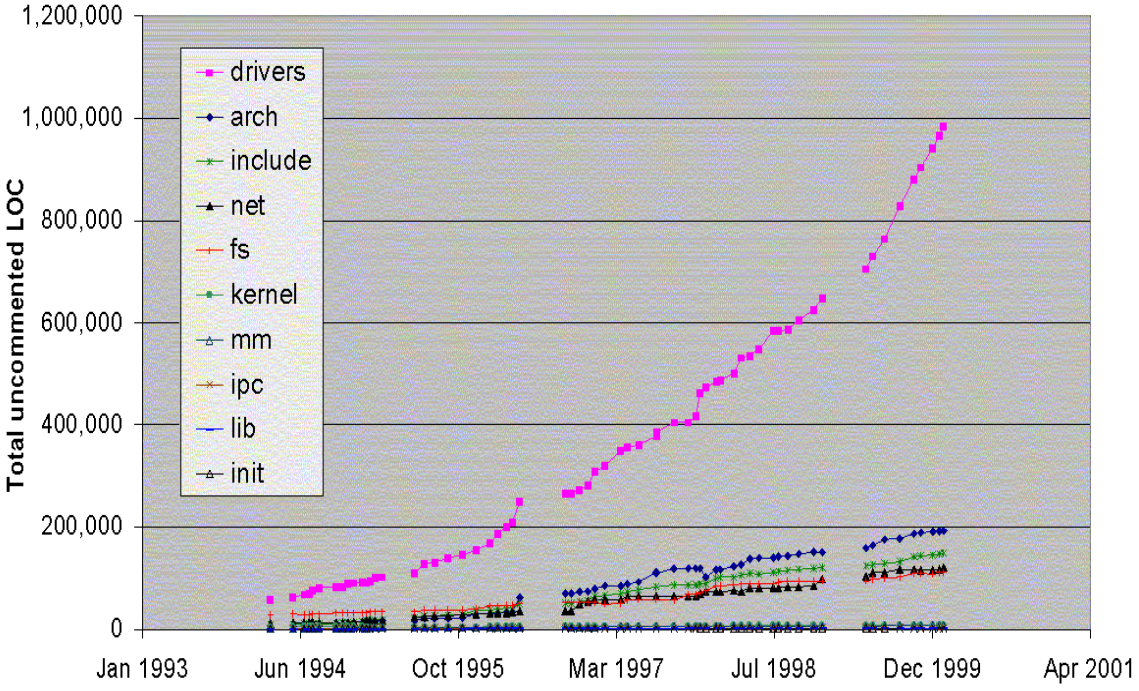


Figure 1. Data revealing the size and growth of major sub-systems in the Linux Kernel during 1994-1999 [Source: Godfrey and Tu 2000].

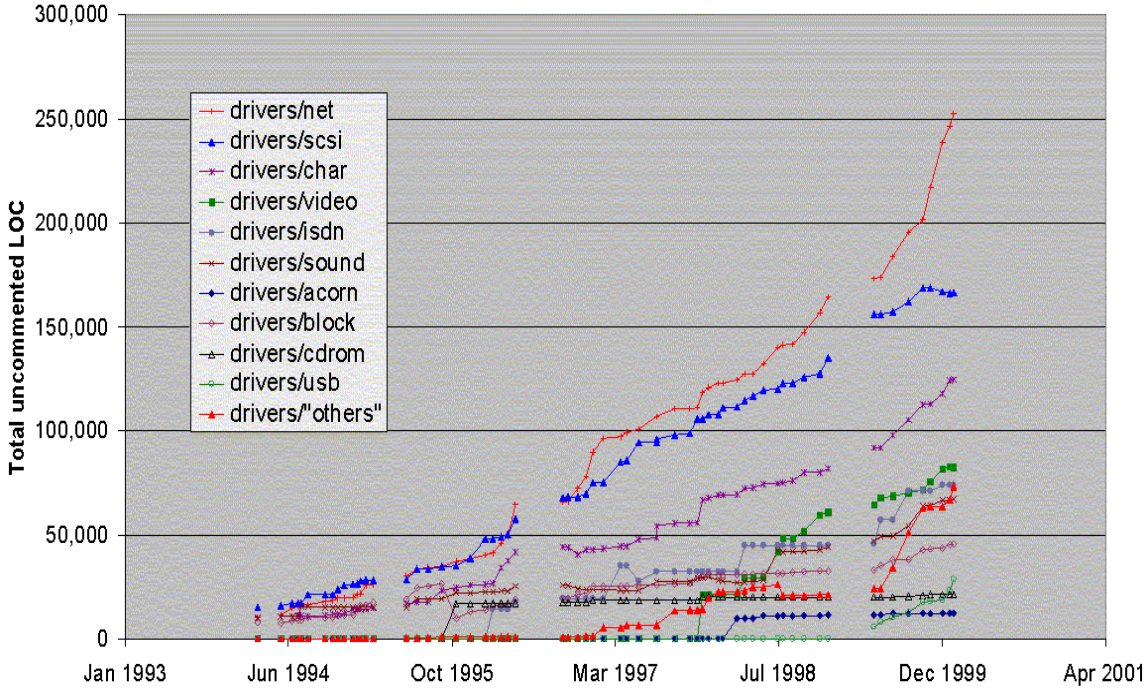


Figure 2. Data revealing the size and growth of device drivers in the Linux Kernel during 1994-1999 [Source: Godfrey and Tu 2000].

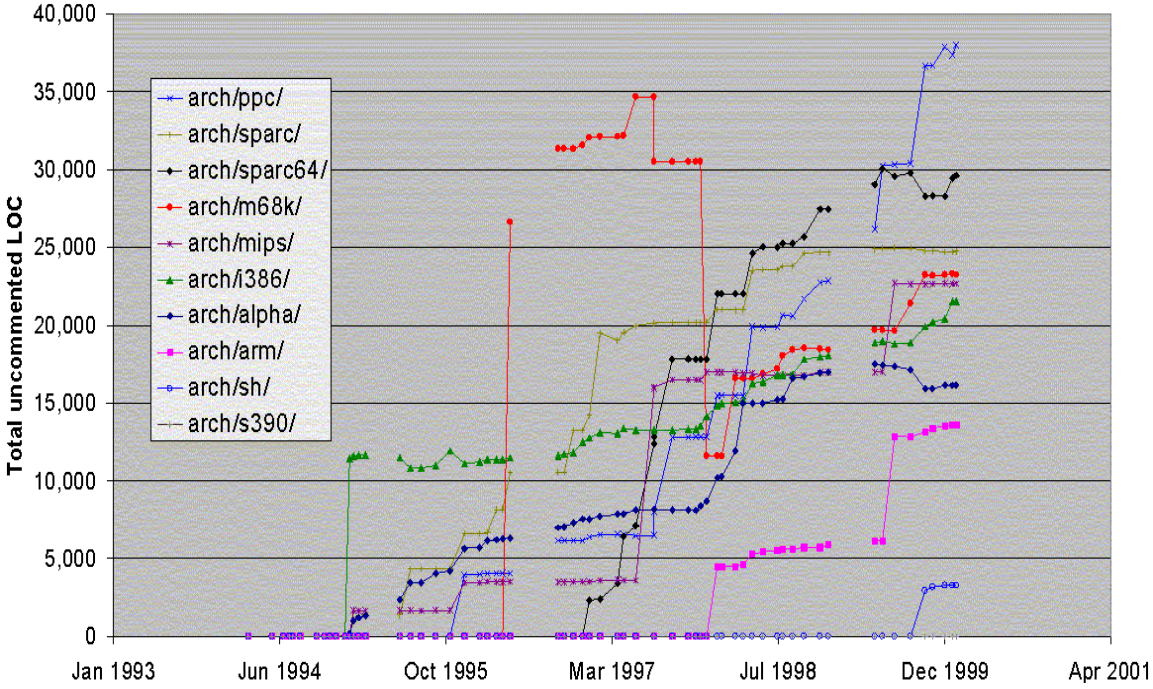


Figure 3. Data revealing the size and growth of the Linux Kernel for different computer platform architectures during 1994-1999 [Source: Godfrey and Tu 2000].

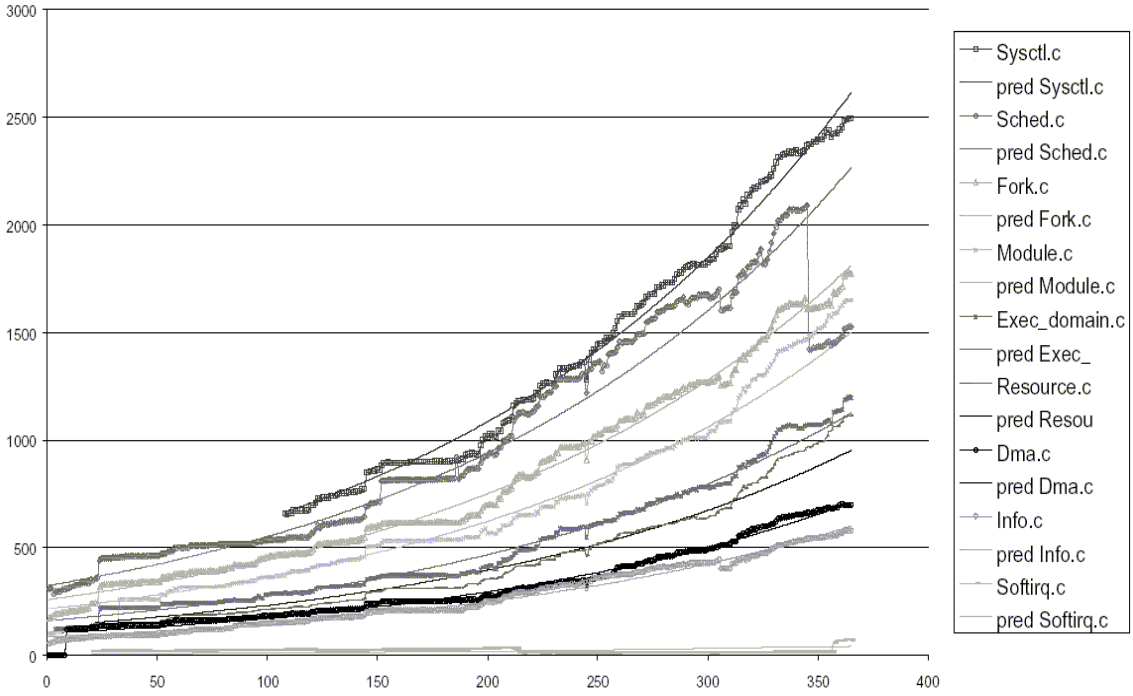


Figure 4. Measured (discrete points) versus predicted (smooth curves) of common coupling of source code modules in the Linux Kernel across releases
[Source: Schach, Jin, *et al*., 2002].

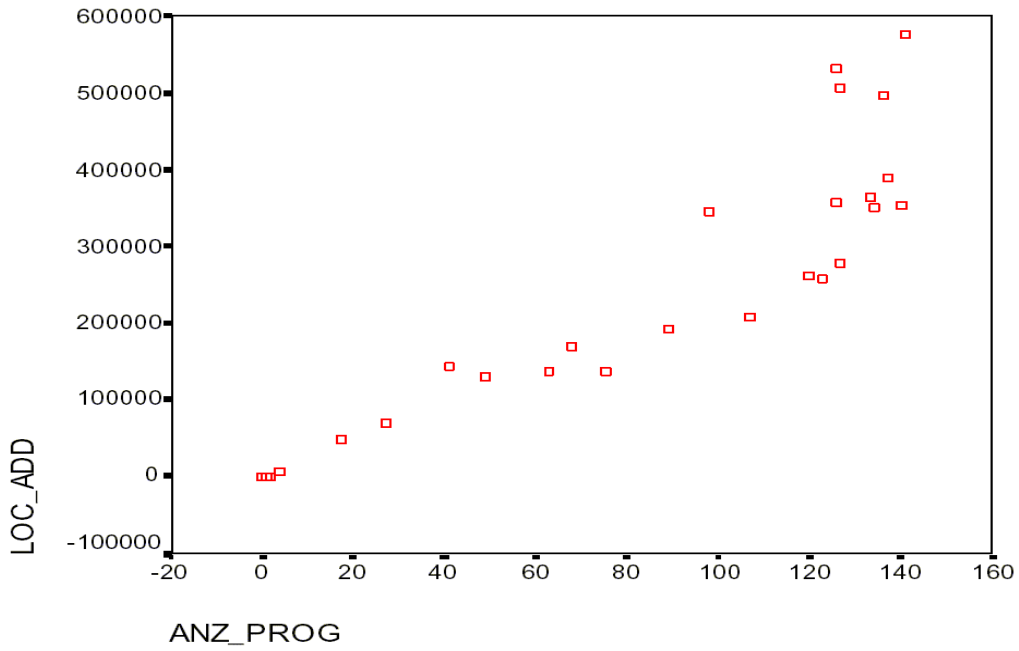


Figure 5. Growth of the lines of source code added as the number of software developers contributing code to the GNOME user interface grows
[Source: Koch and Schneider 2000].

Robles-Martinez, Gonzalez-Barahona, *et al.,* [2003] report in their study of Mono (a F/OSS implementation of Microsoft's .NET services, libraries, and interfaces), their measurements indicate super-linear growth rate in the code size and the number of code updates that are committed within the code base. They also report a similar growth pattern in the number of people contributing source code to the emerging Mono system over a 2-3 year period. According to Gonzalez-Barahona, Ortuno Perez, *et al*, [2001] their measurements indicate that as of mid-2001, the Debian GNU/Linux 2.2 distribution had grown to more than 55M SLOC, and has since exceeded 100M SLOC in the Debian 3.0 distribution. O'Mahony [2003] presents data from her study of the Debian Gnu/Linux distribution from releases spanning 0.01 in 1993 through 3.0 in late 2002 that show growth of the size of the distribution rises at a super-linear rate over the past five years. Last, Gonzalez-Barahona, Lopez, and Robles [2004] also provide data on the growth of the Apache project community and number of modules, revealing once again, a super-linear growth pattern over the five year period (1999-2004) covered in their data.

In contrast, Godfrey and Tu [2000] find linear growth in Fetchmail, X-Windows, and Gcc (the GNU compiler collection), and sub-linear growth in Pine (email client). Such trends are clearly different from the previous set of F/OSS systems.

Why is there such a high growth rate for some F/OSS systems like the Linux Kernel, Vim, GNOME, Mono, the Debian GNU/Linux distribution, and the Apache project, but not for other F/OSS? Godfrey and Tu [2000] report in the case of the Linux Kernel that (a) much of the source code relates to device drivers, as seen in Figure 2, (b) much of the code is orthogonal and intended for different platforms, as suggested in Figure 3, and (c) contributions to the code base are open to anyone who makes the requisite effort. In addition, Godfrey and Tu observe (d) Linux Kernel source code configurations (or "builds") are specific to a hardware platform or architecture (see Figure 3), and use as little of 15% of the total Linux Kernel source code base. It is possible but uncertain whether these conditions also apply to GNOME, Vim, Mono and the Apache project, since they may have source code configurations that are specific to different operating systems (Linux, BSD, Windows, or Mac OS/X). However, it is unclear why they would or would not apply to Fetchmail, X-Windows, Gcc and Pine. Perhaps it might be because the latter systems are generally older and may have originally been developed in an earlier (pre-Web) technological regime. Elsewhere, Cook, Ji, and Harrison [2000] in their comparison study of the closed-source Logica FW system, and the F/OSS Berkeley DB system, find that growth across releases is not uniformly distributed, but concentrated in different system modules across releases. A similar result may be seen in the data in Figure 3, from Godfrey and Tu [2000].

Nakakoji, Yamamoto, *et al*., [2002] report findings from a comparative case study of four F/OSS systems, the Linux Kernel, Postgres DBMS, GNUWingnut, and Jun a 3D graphics library. They provide data indicating that these systems exhibit different evolutionary patterns of splitting and merging their overall system architectures across releases, as shown in Figure 6. Thus it appears that it is necessary to understand both the *age* and *architectural patterns* of sub-systems and modules within and across software releases, whether in closed source or open source systems, in order to better understand how a

system is evolving [Godfrey and Lee 2000]. This observation is also implicated by earlier studies [Tamai and Torimitsu 1992, Gall, Jayazeri, *et al*. 1997, Eick, Graves, *et al.* 2001, Perry, Siy, and Votta 2001].

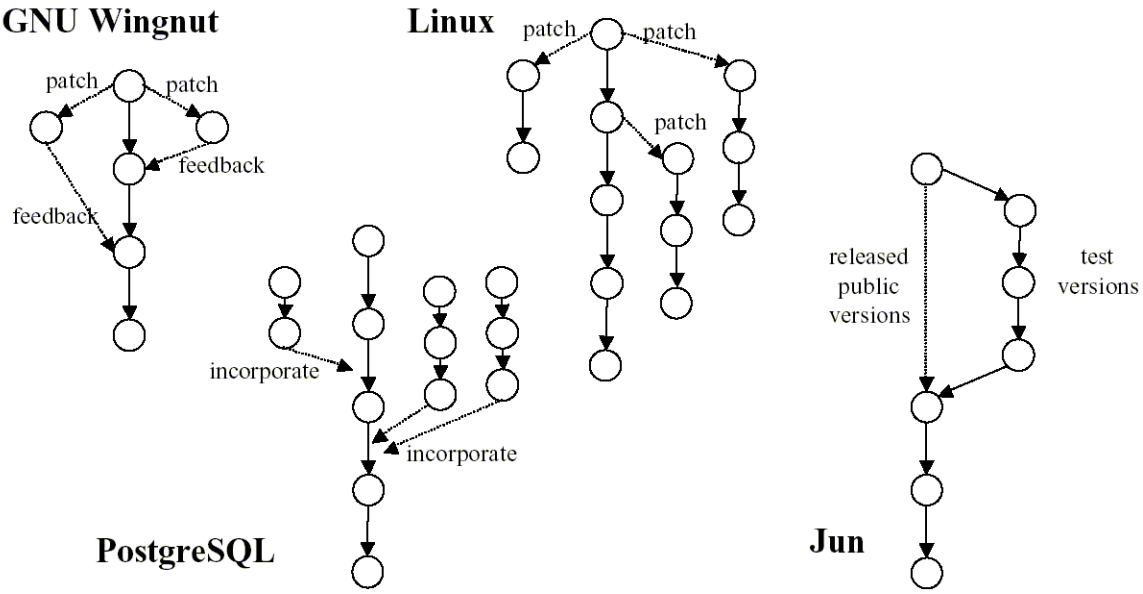


Figure 6. Patterns of software system evolution forking and joining across releases (nodes in each graph) for four different F/OSS systems
[Source: Nakakoji, Yamamoto, *et al*., 2002]

Hunt and Johnson [2002] report discovery of a Pareto distribution in the size of the number of developers participating in F/OSS projects, from a sample population of >30K projects found on the SourceForge Web portal.[5] Their results indicate that the vast majority of F/OSS projects have only one developer, while a small percentage have larger, ongoing team membership. Madey, Freeh, and Tynan [2002] in an independent study similar to Hunt and Johnson, find that a power law distribution characterizes the size of F/OSSD projects across a population of some 40K F/OSS projects at SourceForge. Independently, Hars and Ou [2002] report a similar trend, finding that more than 60 percent of F/OSS developers in their survey reported participating in 2-10 other F/OSS development projects. Capilupppi, Lago, and Morisio [2003] also draw from a sample of 400 F/OSSD projects posted on SourceForge. They find that the vast majority of systems in their sample are either small or medium size systems, and only a minor fraction are large. Only the large F/OSS systems tend to have development teams with more than a single developer. Their results might also be compared to those of Tamai and Torimitsu [1992], thereby substantiating that small F/OSS systems have a much shorter life, compared to large F/OSS systems. Overall, this suggests that results from studies that characterize large F/OSS efforts are not representative of the majority of F/OSS projects.

[5] The SourceForge Web portal can be found at www.sourceforge.net. At the moment, there are 85K F/OSS projects now registered at this specific F/OSS project portal. Other F/OSS Web portals like www.freshmeat.org and www.savannah.org include other projects, though there is some overlap across these three portals.

Di Penta, Neteler, *et al.,* [2002] provide results from a case study focused on the refactoring of a large F/OSS application, a geographical information system called GRASS, which operates on a small hand-held computer. Their effort was aimed at software miniaturization, reducing code duplications, eliminating unused files, and restructuring system libraries and reorganizing them into shared (i.e., dynamically linked) libraries. This form of software evolution and architectural refactoring has not been reported in, or accounted for by, the laws of software evolution. For example, miniaturization and refactoring will reduce the size of the software application, as well as potentially reducing redundancies and code decay, thereby improving software quality. Elsewhere, Scacchi [2002c] reports results from a case study of the GNUenterprise project that find that the emerging F/OSS E-Commerce application system being developed is growing through merger with other independently developed F/OSS systems, none of which was designed or envisioned as a target for merger or component sub-system. He labels this discontinuous growth of F/OSS system size and functionality, *architectural bricolage.* Such capabilities may account for the discontinuities that can be seen in the growth trends displayed in Figure 3.

Mockus, Fielding, Herbsleb [2002] in a comparative case study of Apache Web server (<100K SLOC) and Mozilla Web browser (2M+ SLOC), find that it appears easier to maintain the quality of system features for a F/OSS across releases compared to closed-source commercial telecommunications systems of similar proportions. They also find evidence suggesting large F/OSS development projects must attain a core developer team size of 10-15 developers for its evolution to be sustained. This might thus be recognized as an indicator for *a critical mass in the number of core developers* that once achieved enables a high rate of growth and sustained viability. Whether and how long such growth can be sustained however is unclear as the number of core developers changes over time.

Scacchi and colleagues [2002a, 2002c, Elliott and Scacchi 2002, Jensen and Scacchi 2003] provide results from comparative case studies of F/OSS projects within different communities. They find and explicitly model how F/OSS requirements and release processes differ from those expected in conventional software engineering practices. They also find that evolving F/OSS depends on co-evolution of developer community, community support software, and software informalisms as documentation and communication media. Nakakoji, Yamamoto, *et al*., [2002] also report that the four F/OSS systems they investigated co-evolve with the communities of developers who maintain them. Finally, Gonzalez-Barahona, Lopez, and Robles [2004] provide a detailed data set that visualizes the growth of the developer community over a five year period corresponding to growth in the number of modules incorporated in the Apache project.

Von Hippel and Katz [2002] report results of studies that reveal some end-users in F/OSS projects become developers, and most F/OSS developers are end-users of the systems they develop, thereby enabling the co-evolution of the system and user-developer community. This observation of developers as users as developers is also independently reported in other studies as well [Mockus, Fielding, Herbsleb 2002, Scacchi 2002a, and Nakakoji, Yamamoto, *et al*., 2002]. Last, much like Hars and Ou [2002], Madey, Freeh, and Tynan [2002] report finding that some F/OSS developers, whom they designate as *linchpin developers*, participate in multiple projects. These linchpin developers

effectively create social networks that interlink F/OSS projects and enable the systems interlinked in these social networks to also share source code or sub-systems. A sample from their data appears in Figure 7.

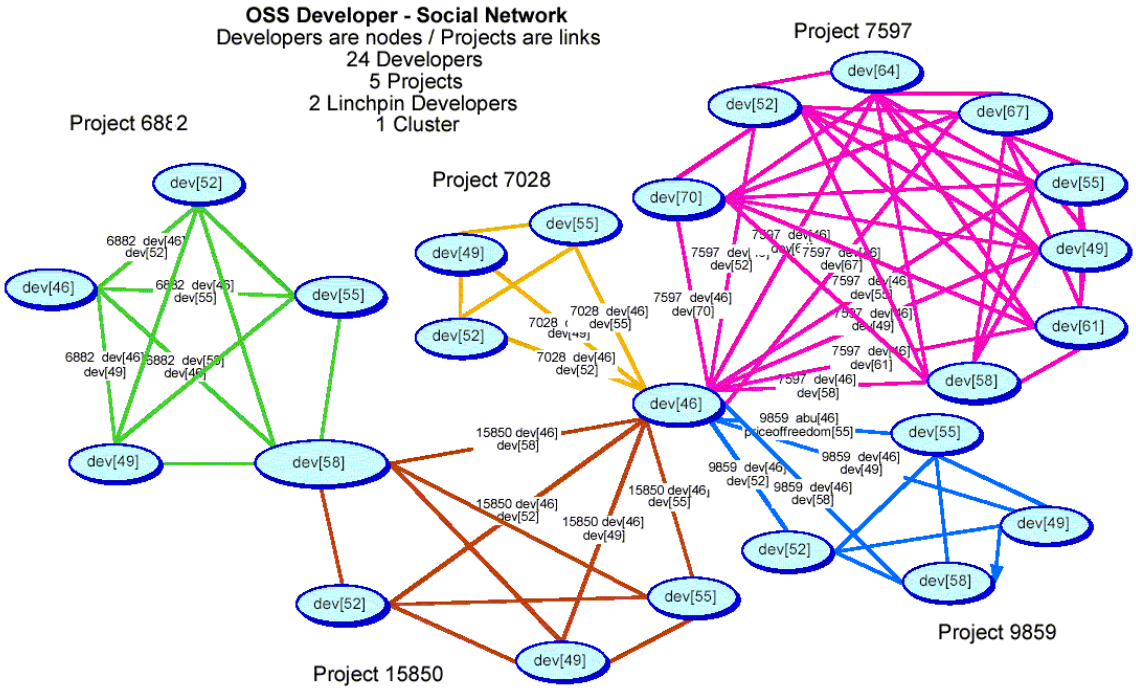


Figure 7. A social network of F/OSS developers that interlinks five different projects through two linchpin developers, dev[46] and dev[58]
[Source: Madey, Freeh, and Tynan 2002].

However, across the set of studies starting above with Mockus, Fielding and Herbsleb [2002], there are no equivalent observations or laws reported in prior studies of closed source software evolution that account for these data and evolutionary patterns addressing team and community structures. Clearly, such a result does not imply that the observed conditions or patterns do not occur in the evolution of closed source software. Instead, it reveals that other variables and patterns previously not addressed in prior empirical studies may be significant factors contributing to software evolution.

Last, in a recent study comparing open versus closed source software development products, Paulson, Succi, and Eberlein [2004] find that overall evolutionary growth of both types of software are comparable, and consistent with the laws of software evolution, for the systems they examined. Specifically, in modeling and visualizing the growth of the systems in their studies, as displayed in Figure 8, their research design employs linear approximations to depict system growth trends over time. Thus, it is not possible to tell from their results if these approximations "linearize" the inverse-square growth curves reported by Lehman and colleagues or the exponential curves of the kind shown in Figures 1 through 5, or the non-linear growth shown in Figure 6. However, they

do plot a growth curve for the Linux as seen in Figure 8, which may suggest their linear approximation in this instance flattens the exponential growth pattern for Linux seen in Figures 1 through 5.

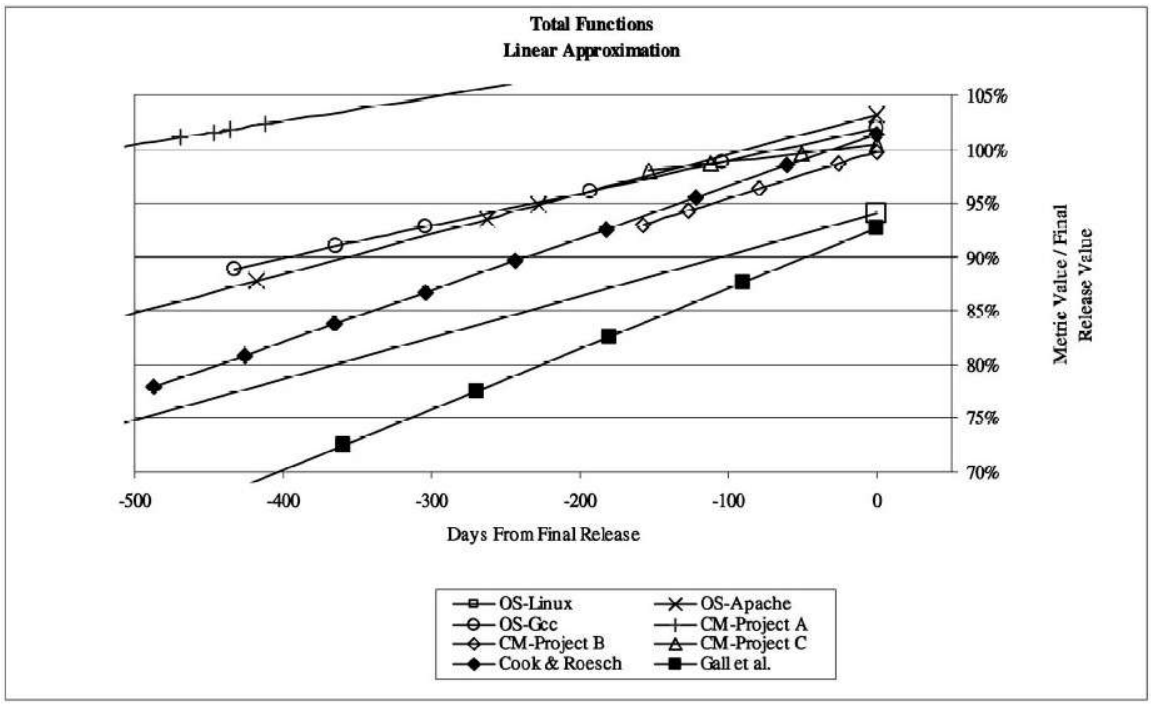


Figure 8. Linear approximations of the growth of a sample of open and closed source software systems [Source: Paulson, Succi, and Eberlein, 2004]

Overall, in evolving large F/OSS, it seems that it may be necessary for a critical mass of developers to come together to anchor the broader community of users, developers, and user-developers to their shared system. This critical mass and community will co-evolve with the architectural patterns that are manifest across unstable and stable F/OSS system releases as they age over time. Subsequently, older F/OSS systems that may have emerged before the F/OSS gained widespread recognition as a social movement and cultural meme, may have a lower rate of architectural and system release co-evolution. Furthermore, it may well be the situation that for large F/OSS systems/releases to evolve at a super-linear rate, that this may be possible only when their development community has critical mass, is open to ongoing growth, and that the focal F/OSS systems entail internal architectures with orthogonal features, sub-systems, or modules, as well as external system release architectures that span multiple deployment platforms.

Last, it appears that the evolutionary patterns of F/OSS systems reveal that overall system size and architecture can increase or decrease in a dis-continuous manner, due to bricolage-style system mergers, or to miniaturization and refactoring. Clearly, the laws of software evolution as presently stated, and based primarily on the study of large closed source systems, do not account for, nor anticipate, the potential for super-linear growth in

software system size that can be sustained in the presence of satisfied developer-user communities who collectively assure the quality of these systems over time.

4. Evolution Models and Theories

As a first step, it is desirable to provide a definition of evolution that is satisfied by examples that cover different scientific and technological disciplines. Lehman and Ramil [2004] provide an appropriate definition for software evolution. In the definition that follows, an attempt has been made to address properties applicable in a general sense. Individual disciplines may have additional properties not identified here. Accordingly, evolution is a process of progressive change and cyclic adaptation over time in terms of the attributes, behavioral properties, and relational configuration of some material, abstract, natural or artificial entity or system. Such a definition accommodates both "evolutionistic" models that draw attention to stages and direction of developmental progress, and "evolutionary" models that focus attention to mechanisms, forces, or impinging constraints that give rise to evolution [cf. King and Kraemer 1984].

Theories of biological evolution have been the subject of scientific inquiry and speculation for centuries, with Charles Darwin's *Origin of Species* (1859) being the most widely known and cited theory. Darwin's theoretical analysis was based in part on his field studies of animal species on the Galapagos archipelago. His theory begat a century of scientific debate that included religious and moral substance undertones [Bowler 1989]. It led to the emergence of concepts such as *developmental biology* that examines the role of genetics, reproductive (natural) selection, co-evolution (among co-located species) and adaptation to ecological circumstances shaping the lives of organisms. In contrast, *evolutionary biology* accounts for the influence of genetics, reproduction, lineage, speciation, and population growth and diffusion shaping the long-term or trans-generational lives of species of organisms. The concepts of developmental versus evolutionary biology help draw attention to two alternative ways to view the evolution of a system, one focusing on a system's life cycle, the other on changes manifest across generations of related systems. In addition, the concept of biological *systematics* further helps draw attention to the "progress", direction, or (punctuated) equilibrium of evolution based on associating the developmental properties (e.g., agency, efficiency, and scope) of living organisms with those found in the fossil and geological record [Nitecki 1988, Gould 2002].

Culture, language and economy, which arise from the social actions and interactions of people, may evolve in ways similar or different from that in the natural science of biology. Culture, for example, may rely on the development, diffusion, and assimilation of *memes* (i.e., concepts, compelling ideas, or cultural "genes") that embody recurring social practices, situations, beliefs, or myths that can be shared, communicated, or otherwise transported as a basis for their evolution [Gabora 1997]. Despite differences, "open source" and "free software" are examples of related memes. Thus, rather than conjecturing physical (biological) conditions or circumstances, cultural evolution relies on social actions and narrative records that relate to physical conditions and circumstances that enable the ongoing evolution of diverse cultures and cultural

experiences. Language evolution [Christiansen and Kirby 2003] seems to share and span ideas from culture and biology with respect to efforts that associate language learning, perception, and semiotics with neurological mechanisms and human population dynamics. Elsewhere, topics like *competition*, *resource scarcity*, *population concentration/density*, *legitimacy*, and *organizational ecologies* appear as factors shaping the evolution of markets, organizations and economies, at least at a macro level [Hannan and Carroll 1992, Nelson and Winter 1982, Saviotti and Mani 1995]. Beyond this, the evolution of culture, language and economy are being explored experimentally using computational approaches [e.g., Gabora 2000]. Overall, this tiny sample of work draws attention to associations more closely aligned to evolutionary biology, rather than to developmental biology.

The evolution of modern technology has also become the subject of systematic inquiry. For example, in Abernathy's [1978] study of the American automobile industry, he finds that the *technical system* for developing and manufacturing automobiles associates product design and process design within a *productive unit* (i.e., the manufacturing systems within a physical factory or production organization). Each depends on the other, so that changes in one, such as the introduction of new techniques into a productive unit, are propagated into both product design and production process layout/workflow. King and Kraemer [1984] provide similar findings in their analysis of the evolution of computing systems in organizational settings. Hughes [1987] in his historical study of the technical system of electrification draws attention to the role of the *infrastructure* of electrical production and distribution as spanning not just equipment, mechanisms (e.g., power generators, sub-stations), cabling and power outlets, but also the *alignment* of producers, retailers, and consumers of devices/products together with the processes that depend on electrification for their operation. Meyer and Utterback [1994] were among the first to recognize that productive units and technical systems of production and consumption were increasingly organized around *product lines* that accommodate a diversity of product life cycles centered around the *dominant design* [Utterback 1994] or product architecture that dominates current retail markets.

From an economic perspective, Nelson and Winter [1982] independently termed the overall scheme that associates and aligns products, processes, productive units with producers, retailers and consumers, a *technological regime*. Last, though the development, use, and maintenance of software is strongly dependent on computer hardware, there are now studies that examine how different kinds of computer hardware components exhibit evolutionary patterns across technological generations or regimes [e.g., Victor and Ausubel 2002, van de Ende and Kemp 1999]. The evolution of technology through technological regimes that depend on product features, development processes, infrastructure, and productive units seems immediately relevant to understanding the evolution of software systems and technologies.

Software programs, systems, applications, processes, and productive units continue to develop over time. For example, there is a plethora of software innovations in the form of new tools, techniques, concepts or applications, which continue to emerge as more people experience modern computing technology and technical systems. These innovations give rise to unexpected or unanticipated forms of software development and maintenance, as,

for example, software systems that are dynamically linked at run-time instead of compile-time [Mens *et al.*, 2003, Kniesel *et al.*, 2002]. Software innovations are diffused into a population of evermore diverse settings and technical systems via technology transfer and system migration. Software processes are subject to ongoing experience, learning, improvement and refinement, though there is debate about how to most effectively and efficiently realize and assimilate such process improvements [Conradi and Fuggetta 2002, Beecham, Hall and Rainer 2003]. Software systems are also subject to cultural forces [Elliott and Scacchi 2002], narrative and informal documentation [Scacchi 2002a] and economic conditions [Boehm 1981] within the productive units or work settings that affect how these systems will be developed and maintained. These forces can give rise to similar kinds of systems in similar settings evolving at different rates along different trajectories [Bendifallah and Scacchi 1987]. This suggests that software systems are developed and evolved within particular organizational and informational ecologies [cf. Nardi and O'Day 1999], as well as situated within a technical system of production and larger overall technological regime.

Overall, this brief review of evolutionary theory across a sample of disciplines raises an awareness of the following issues. First, in studying software evolution, it is necessary to clarify whether in fact attention is directed at matters more closely aligned to development of a given system throughout its life, or with the evolution of software technologies across generations that are disseminated across multiple populations. It appears that much of what are labeled as studies of "software evolution" are more typically studies of patterns of development of specific systems, rather than patterns of evolution across different systems within one or multiple product lines (or species), at least as compared to work in biological evolution. However, the laws and theory of software evolution articulated by Lehman and associates depend on empirical findings that examine a variety of software systems in different application domains, execution environments, size of system, organization, and company marketing the system, as their basis for identifying mechanisms and conditions that affect software evolution.

Second, when considering the subject of software evolution at a macro level, it appears that there are no easily found or widely cited studies of that examine issues of memes, competition, resource scarcity, population concentration/density, legitimacy, and organizational ecology as forces that shape or impinge on software systems or software technology. The study of software evolution is still in its infancy. In general, existing theory of the development or evolution of software does not yet have a substantial cultural, language, or economic basis. Studies, analyses, and insights from these arenas are yet to appear, and thus need to be explored.

Last, conventional closed source software systems developed within centralized corporate productive units and open source software systems developed within globally decentralized settings without corporate locale represent alternative technological regimes. Each represents a different technical system of production, distribution/retailing, consumption, differentiated product lines, dominant product designs and more. Similarly, software development methods based on object-oriented design and coding, agile development, and extreme programming entail some form of alternative technological regime. Concepts from theories of technological evolution and observations on patterns

of software development and maintenance can be used to help shape an understanding of how software evolves. Additional work is required to compare and contrast evolutionary behavior under different software development regimes. The present discussion concentrates on the socio-technical regime of open source software development.

5. Do We Need New or Revised Models, Laws, or Theories for Open Source Software Evolution?

At this point it is reasonable to ask about whether prior studies, models, or laws adequately account for the evolution F/OSS systems, at least according to the studies and data presented above. For example, other studies of software evolution do not provide a suitable account for the sometimes super-linear, sometimes sub-linear growth curves reported in studies and figures of F/OSS presented above. Beyond this, data trends and patterns accounting for the evolution of F/OSS in some cases conform, and in other cases it is unclear whether or how different F/OSS conform to the laws of software evolution. As such, refining or reformulating them to account for the data at hand is beyond the scope of this chapter. However, it is possible to consider the underlying ontologies for software evolution to rethink what kinds of theory or models of software evolution may further help in understanding, reasoning about, and explaining the evolution of both closed and open source software systems.

5.1 Embracing the Feedback Control Systems Ontology

Feedback and feedback systems appear to be a central part of the conceptual foundation of the laws and theory of software evolution developed by Lehman and colleagues. So why not refine these laws in a way that more fully embraces feedback control theory in articulating the laws so that they can address the evolution of F/OSS? The system dynamics modeling and simulation approach has been widely used to study software project management [Abdel-Hamid and Madnick 1991] and various types of software processes. However, the approach can also be used to model and simulate feedback control systems expressed as a system of differential equations [Bateson 1993, Dolye, Francis, and Tannenbaum 1992]. Combining the laws of software evolution with modeling concepts from system dynamics and feedback control systems should be possible with an eye toward the interests of the audience for whom the laws are intended to serve. For example, executives and senior managers responsible for large software development centers want to know where to make strategic investments and how to better allocate their software development staff, schedules, and related resources. Software developers may want to know what kinds of tools and techniques to use to make their software evolution efforts faster, better and cheaper. Scholars of software evolution theory want to know what kinds of data to collect, what types and sizes of systems to study, what types of criteria to use in designing theoretically motivated samples of software systems under study, and what tests to apply to verify, refine, or refute the laws or theory at hand. In terms of feedback control systems, there is need to identify where sensors should be placed in a software productive unit to collect different types of software change data, and to what or whom they should provide their feed back.

Similarly, there is need to identify where feedback control loops are to be placed, where their begin and end points are to be located, what functionality is located

within each loop, and what decision function determines whether a loop iterates or exits. It is also necessary to identify what roles people and software tools play in the regulation or control of the feedback system, or what feedback they produce, use, or consume along the way. Managers, developers, and scholars want to know how different types of feedback get employed to regulate and control the centralized corporate or decentralized open source productive unit that develops and maintains software systems of different size, type, age, and application setting.

Recent efforts by Lehman, Ramil and Kahen [2001], for example, have begun to employ system dynamics modeling techniques and simulation tools to demonstrate and iteratively refine an operational model of software evolution that embodies the existing laws. Their model seems able to reproduce via simulation the evolutionary data trends that conform to the laws of software evolution. More recently, Ramil and colleagues [Smith, Capiluppi, and Ramil 2004] examine and qualitatively simulate F/OSS evolution data for 26 systems they have studied. Their data and analyses from this latest study confirm and incorporate further refinements to the laws of software evolution, though they also find some puzzling trends that are not well explained by the laws. However, the trends reported in their data appear to differ from many of the studies prior those published before 2000 that gave rise to the laws of software evolution. But the stage is set for how to proceed in pursuing the ontological foundation of the laws and theory of software evolution.

On the other hand, if the theory of feedback control systems becomes too complicated or too rigid of an ontological framework for describing and explaining software evolution, then alternative ontological frameworks may be employed to further such study.

5.2 Alternative Ontologies for F/OSS Evolution

One observation from studying the evolution of technical systems is that the technologies and techniques for developing and maintaining F/OSS constitute a distinct technological regime. This regime for F/OSS is not anticipated or adequately covered by other studies of software evolution. The same may also be true of emerging technologies like component-based software systems and those with dynamically composed run-time architectures. Thus, it seems that any ontology for software evolution should account for the emergence, deployment, and consequences of use for new tools, techniques, and concepts for software development, as well as the productive units, technical system infrastructure, and technological regime in they are situated.

A second observation from the study of the evolution of F/OSS is that different types of software system evolve at substantially different rates--some super-linear, some constant, some sub-linear, some not at all. Small software systems may not evolve or thrive for very long, nor will they be assimilated into larger systems, unless merged with other systems whose developers can form a critical mass sufficient to co-evolve with the composite system and productive unit. Drawing from biological evolutionary theory, it may be that software evolution theory requires or will benefit from *taxonomic analyses* to describe, classify and name different types of software systems or architectural morphologies, thus refactoring the conceptual space for software system evolution. Similarly, it may benefit from *phylogenetic analyses* that reconstruct the evolutionary histories of different types of software systems, whether as open source and closed source

implementations. Last, it suggests that a science of *software systematics* is needed to encourage study of the kinds and diversity of software programs, components, systems, and application domains, as well as relationships among them, across populations of development projects within different technological regimes over time. This would enable comparative study of contemporary software systems with their ancestral lineage, as well as to those found within the software fossil record (e.g., those software systems developed starting in the 1940's onward for mainframe computers, and those developed starting in the 1970's for personal computers). Finally, this could all be done in ways that enable free/open source computational modeling of such a framework for software evolution.

A third observation from the emergence and evolution of F/OSS is that the beliefs, narratives, and memes play a role in facilitating the adoption, deployment, use and evolution of F/OSS. Their role may be more significant than the cultural and language constructs that accompanied the earlier technological regime of centralized, closed source software development that primarily developed systems for deployment in corporate settings. Similarly, relatively new software language constructs for scripting, plug-in modules, and extensible software architectures have been popularized in the regime of F/OSS. But these constructs may also have enabled new forms of architectural evolution and bricolage, thereby accelerating the growth rate of large F/OSS in a manner incommensurate to that seen in the world of mainframe software systems, an earlier technological regime. Finally, large and popular F/OSS systems are being extended and evolved to accommodate end-users and developers whose native language or ethnic legacy is not English based. The internationalization or localization of F/OSS systems, while neither necessarily adding nor subtracting functionality, does create value in the global community by making these systems more accessible to a larger audience of prospective end-users, developers, reviewers and debuggers. These software extensions add to the bulk of F/OSS code release size in probably orthogonal ways, but may or may not represent anti-regressive work [cf. Lehman, Ramil, and Kahan 2001].

A fourth observation from the evolution of F/OSS is that they have emerged within a technological regime where competitive market forces and organizational ecologies surrounding closed source software systems may have effectively served to stimulate the growth and diffusion of F/OSS project populations. Furthermore, it may be the case that these circumstances are co-evolving with the relative growth/demise of open versus closed source software product offerings, and the communities of developers who support them. Other studies of software evolution make little/no statement about the effects of market forces, competition, organizational ecology, co-evolution, or the spread of software project populations as contributing factors affecting how software systems may evolve. Yet many of the largest F/OSS systems are pitted directly against commercially available, closed source alternatives. These F/OSS systems typically compete against those developed within centrally controlled and resource managed software development centers. Thus, it seems appropriate to address how co-evolutionary market forces surround and situate the centralized or decentralized organizational ecologies that develop and maintain large software systems in order to better understand how they evolve.

A last observation from a view of F/OSS as a socio-technical world is that the evolution of F/OSS system is situated within distinct web of organizational, technological, historical and geographic contexts. However, feedback control systems typically do not account for organizational productive units or their historical circumstances. Similarly, there is no accounting for the motivations, beliefs, or cultural values of software developers who may prefer software systems to be developed in a manner that is free and open, so as to enable subsequent study, learning, reinvention, modification, and redistribution. But as seen above, these are plausible variables that can contribute to the evolution of F/OSS, and thus further study is required to understand when, where and how they might influence how particular F/OSS systems may evolve.

6. Conclusions

The laws and theory of software evolution proposed by Lehman and colleagues are recognized as a major contribution to the field of software engineering and the discipline of computer science. These laws have been generally found to provide a plausible explanation for how software systems evolve throughout their life. They have been explored empirically over a period of more than 30 years, so their persistence is a noteworthy accomplishment. Developing laws and theory of software evolution relying on empirically grounded studies is a long-term endeavor that poses many challenges in research method, theoretical sampling of systems to study, theory construction, and ongoing theory testing, refutation, and refinement. However, it may prove to be an endeavor that gives rise to new ways and means for conceptualizing evolutionary processes in other domains of study.

As the technology, process, and practice of software development and maintenance has evolved, particularly in the past ten years and with the advent of large numbers of free/open source software development projects, it has begun clear that the existing models of software evolution based on empirical studies of closed source systems prior to 2000 may be breaking down, at least from results of the many empirical studies of F/OSS reviewed in this chapter. The models and prior studies do not address and therefore do not provide a rich or deep characterization of the evolution of F/OSS systems. Prior models of software evolution were formulated in the context of software development and maintenance processes and work practices that were based in centralized, corporate software development centers that built large closed source system applications with few competitive offerings for use by large enterprises. Large F/OSS systems, on the other hand, are developed and maintained in globally decentralized settings that collectively denote a loosely-coupled community of developers/users who generally lack the administrative authority, resource constraints, and schedules found in centrally controlled software centers. These F/OSS systems are typically competing alternatives to closed source commercial software product offerings. Subsequently, it may be better to consider whether the primary evolutionary dynamic associated with F/OSS is reinvention, renovation, or revitalization of established software systems or applications that have proved to be useful, but now merit redevelopment, refinement, and new extensions or extension mechanisms [Scacchi 2004]. Similarly, as large F/OSS are sometimes observed to exhibit sustained super-linear or exponential growth, can such rates of growth go on unabated, or will the concurrent growth of system complexity eventual change the shape

of the growth curve to something more like an "S" curve, with exponential growth in the early stages, followed by inverse-square growth in the later stages [cf. Lehman and Ramil 2002]. Further study of such matters is clearly needed.

There is a growing base of data, evidence, and findings from multiple studies of F/OSS systems that indicate F/OSS systems co-evolve with their user-developer communities, so that growth and evolution of each depends on the other. Co-evolution results of this kind are not yet reported for closed source systems, and it is unclear that such results will be found. In short, prior models of software evolution were developed within and apply to systems maintained and used in a corporate world and technological regime that differs from the socio-technical communities, global information infrastructure, and technological regime which embeds open source software.

It appears that we need a more articulate explication and refinement of models of software evolution if they are to account for the evolution of F/OSS systems. One way this might be done is to embrace and extend reliance of the ontology of feedback control systems theory. This would entail identifying the types, operations, behaviors, and interconnection of mechanisms that embody and realize a complex, multi-level, multi-loop, and multi-agent feedback system. Building computational models and simulations of such a system (or family of systems) could be a significant contribution. Otherwise, alternative evolutionary ontologies might be adopted, individually or in some explicit hybrid combination form. The choice of which ontology to use will suggest the types of entities, flows, mechanisms, and controls for software evolution should be modeled, measured, improved, and refined according to some conceptual or theoretically motivated framework. Otherwise, use of alternative ontologies may accommodate new models of theories of software evolution that do not rely on high-level, abstract or over-generalized models, but instead may result in theories or models of smaller and more precise scope that better account for the complex, socio-technical ecological niches where software systems evolve in practice, as well as for the type and history of the system in such context.

Theories of software evolution should be empirically grounded. They should be formulated or modeled in ways in which they can be subject to tests of refutation or refinement. The tests in turn should examine comparative data sets that are theoretically motivated, rather than motivated by the convenience of data at hand that may have been collected and conceived for other more modest purposes. There should be theories that address software evolution within, as well as, across generations of software technology or technological regimes. Laws and theories of software evolution should have a computational rendering so that their source code, internal representation, and external behavior can be observed, shared, studied, modified and redistributed. They should be free (as in *libre*) and open source. These models should then also be suitable for simulation, analysis, visualization, prototyping, and enactment [Scacchi 2002b, Scacchi and Mi 1997]. By doing this, the software engineering and computer science community can make a new contribution in the form of reusable assets that can be adopted and tailored for use in other domains of evolution theorizing.

The future of research in software evolution is must include the technological regime of F/OSS as a major element. This will be an increasingly practical choice for empirical study of individual systems, groups of systems of common type, and of larger regional or global populations of systems. This is due in part to the public availability of the source code and related assets on the Web for individual versions/releases of hundreds of application systems, as well as data about their development processes, community participants, tools in use, and settings of development work. Not that collecting or accessing this data is without its demands for time, skill, effort and therefore cost, but that useful and interesting data can be accessed and shared without the barriers to entry and corporate disclosure constraints of intellectual property claims or trade secrets. It seems unlikely that the software engineering community will get open access to the source code, bug report databases, release histories, or other "property or secrets" of closed source systems that are in widespread use (e.g., Microsoft Windows operating systems, Internet Explorer, Word, Outlook, Office, Oracle DBMS, or SAP R/3) in ways that can be shared and studied without corporate trade secrets, non-disclosure agreements and publication constraints. In contrast, it is possible today to empirically study the ongoing evolution of the GNU/Linux operating systems (Kernel or alternative distributions), the Mozilla Web browser, Open Office, SAP DB, Apache project, or GNUenterprise, which together with their respective technically and socially networked communities, have publicly accessible Web portals and software assets that can be shared, studied, and redistributed to support research into models, laws, and theory of software evolution. The future of research in software evolution should be free, open and constructive, since it will likely take a community of investigators to help make substantial progress in developing, refining, sharing, and publishing models, laws, and theories of software evolution.

7. Acknowledgements

The research described in this report is supported by grants from the National Science Foundation #0083075, #0205679, #0205724 and #0350754. No endorsement implied. Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; Margaret Elliott, Mark Bergman, Chris Jensen and Xiaobin Li at the UCI Institute for Software Research; and Julia Watson at The Ohio State University are also collaborators on the research project from which this article was derived. Finally, Manny Lehman, Nazim Madhavji, and Juan Ramil provided many helpful comments, suggestions, and clarifications on earlier versions of this chapter.

8. References

W.J. Abernathy, *The Productivity Dilemma: Roadblock to Innovation in the Automobile Industry*, John Hopkins University Press, 1978.

T. Abdel-Hamid and S. E. Madnick, *Software Project Dynamics: An Integrated Approach*. Prentice Hall Software Series, New Jersey, 1991.

R.N. Bateson, *Introduction to Control System Technology*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

A. Bauer and M. Pizka, The Contribution of Free Software to Software Evolution, *Proc. Sixth International Workshop on Principles of Software Evolution (IWPSE'03),* 170-179, Helsinki, Finland, September 2003.

S. Beecham, T. Hall, and A. Rainer, Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis, *Empirical Software Engineering*, 8(1), 7-42, 2003.

S. Bendifallah and W. Scacchi, Understanding Software Maintenance Work, *IEEE Trans. Software Engineering*, 13(3), 311-323, March 1987. Reprinted in D. Longstreet (ed.), *Tutorial on Software Maintenance and Computers*, IEEE Computer Society, 1990.

B.E. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

P.J. Bowler, *Evolution: The History of an Idea* (Revised Edition), University of California Press, Berkeley, CA, 1989.

A. Capilupppi, P. Lago, and M. Morisio, Characteristics of Open Source Projects, *Proc. 7th European Conference on Software Maintenance and Reengineering*, March 2003

M. Christiansen and S. Kirby (eds.), *Language Evolution: The States of the Art*, Oxford University Press, 2003.

R. Conradi and A. Fuggetta, Improving Software Process Improvement, *IEEE Software*, 92-99, July-August 2002.

S. Cook, H. Ji and R. Harrison, Software Evolution and Software Evolvability, unpublished manuscript, University of Reading, UK, 2000.

K. Crowston, H. Annabi, and J. Howison, Defining Open Source Software Project Success, *Proc. Intern. Conf. Information Systems (ICIS 2003)*, Seattle, WA, 327-340, December, 2003.

M.A. Cusumano and D.B. Yoffie, Software Development on Internet Time, *Computer*, 60-70, October 1999.

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

M. Di Penta, M. Neteler, G. Antonio, and E. Merlo, Knowledge-Based Library Refactoring for an Open Source Project, *Proc. IEEE Working Conf. Reverse Engineering*, Richmond VA, October 2002.

J.C. Doyle, B.A Francis and A.R. Tannenbaum, *Feedback Control Theory*, Macmillan, New York, 1992.

J. Erenkrantz, Release Management within Open Source Projects, *Proc. 3rd. Workshop on Open Source Software Engineering,* 25th Intern. Conf. Software Engineering, Portland, OR, May 2003.

S.G. Eick, T.L. Graves, A.F. Karr, J.S. Marron, and A. Mockus, Does Code Decay? Assessing the Evidence from Change Management Data, *IEEE Trans. Software Engineering*, 27(1), 1-12, January 2001.

M. Elliott and W. Scacchi, *Free Software Development: Cooperation and Conflict in a Virtual Organizational Culture*, in S. Koch (ed.), *Free/Open Source Software Development*, Idea Press, 2004 (to appear).

L. Gabora, The Origin and Evolution of Culture and Creativity, *Journal of Memetics - Evolutionary Models of Information Transmission*, 1, 1997. http://jom-emit.cfpm.org/vol1/gabora_l.html

L. Gabora, The Beer Can Theory of Creativity, in P. Bentley and D. Corne (eds.) *Creative Evolutionary Systems*, Morgan Kaufman, 2000.

H. Gall, M. Jayazeri, R. Kloesch and G. Trausmuth, Software Evolution Observations Based on Product Release History, *Proc. 1997 Intern. Conf. Software Maintenance (ICSM'97),* Bari, IT, October 1997.

B. Glaser and A. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine Publishing, Chicago, IL, 1976.

M.W. Godfrey and E.H.S. Lee, Secrets from the Monster: Extracting Mozilla's Software Architecture, *Proc. Second Intern. Symp. Constructing Software Engineering Tools* (CoSET-00), Limerick, Ireland, June 2000.

M.W. Godfrey and Q. Tu, Evolution in Open Source Software: A Case Study, *Proc. 2000 Intern. Conf. Software Maintenance* (ICSM-00), San Jose, California, October 2000.

J.M Gonzalez-Barahona, L. Lopez, and G. Robles, Community Structure of modules in the Apache project, *Proc. 4th Workshop on Open Source Software Engineering*, Edinburgh, Scotland, May 2004.

J.M Gonzalez-Barahona, M.A. Ortuno Perez, P. de las Heras Quiros, J. Centeno Gonzalez, and V. Matellan Olivera, Counting Patotoes: The Size of Debian 2.2, *Upgrade Magazine*, II(6), 60-66, December 2001.

S.J. Gould, *The Structure of Evolutionary Theory*, Harvard University Press, Cambridge, MA, 2002.

M.T. Hannan and G.R. Carroll, *Dynamics of Organizational Populations: Density, Legitimation and Competition*, Oxford University Press, New York, 1992.

A. Hars and S. Ou, Working for Free? Motivations for Participating in Open-Source Software Projects, *Intern. J. Electronic Commerce*, 6(3), 25-39, 2002.

T.J. Hughes, The Evolution of Large Technological Systems, in W. Bijker, T. Hughes, and T. Pinch (eds.), *The Social Construction of Technological Systems*, MIT Press, Cambridge, MA, 51-82, 1987.

F. Hunt and P. Johnson, On the Pareto Distribution of SourceForge Projects, in C. Gacek and B. Arief (eds.), *Proc. Open Source Software Development Workshop*, 122-129, Newcastle, UK, February 2002.

C. Jensen and W. Scacchi, Simulating an Automated Approach to Discovery and Modeling of Open Source Software Development Processes, *Proc. 4th Software Process Simulation and Modeling Workshop (ProSim'03),* Portland, OR, May 2003.

C. Jensen and W. Scacchi, Process Modeling Across the Web Information Infrastructure, *Proc. 5th Software Process Simulation and Modeling Workshop (ProSim'04),* Edinburgh, Scotland, UK, May 2004.

J.L. King and K.L. Kraemer, Evolution and Organizational Information Systems: An Assessment of Nolan's Stage Model, *Communications ACM*, 27(5), 466-475, 1984.

C.F. Kemerer and S. Slaughter, An Empirical Approach to Studying Software Evolution, *IEEE Trans. Software Engineering*, 25(4), 493-505, 1999.

G. Kniesel, J. Noppen, T. Mens, and J. Buckley, *WS 9. The First International Workshop on Unanticipated Software Evolution,* Workshop Report, Malaga, Spain, June 2002. http://joint.org/use2002/ecoopWsReportUSE2002.pdf

S. Koch and G. Schneider, Results from Software Engineering Research into Open Source Development Projects Using Public Data, *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft*, Hans R. Hansen und Wolfgang H. Janko (Hrsg.), Nr. 22, Wirtschaftsuniversität Wien, 2000.

I. Lakatos, *Proofs and Refutations: The Logic of Mathematical Discovery*, Cambridge University Press, Cambridge, UK, 1976.

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

M.M. Lehman, Rules and Tools for Software Evolution Planning and Management, in J. Ramil (ed.), *Proc. FEAST 2000*, Imperial College of Science and Technology, London, 53-68, 2000. Also appears with J.F. Ramil in an expanded version as "Rules and Tools for Software Evolution Management," in *Annals of Software Engineering,* 11, 16-44, 2001.

M.M. Lehman, 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.

M.M. Lehman and L.A. Belady, *Program Evolution – Processes of Software Change*, Academic Press, London, 1985.

M.M. Lehman, D.E. Perry and J.F. Ramil, Implications for Evolution Metrics on Software Maintenance, *Proc. 1998 Intern. Conf. Software Maintenance (ICSM'98),* Bethesda, MD, 1998.

M.M. Lehman and J.F. Ramil, An Approach to a Theory of Software Evolution, *Proc. 2001 Intern. Workshop on Principles of Software Evolution*, 2001.

M.M. Lehman and J.F. Ramil, An Overview of Some Lessons Learnt in FEAST, *Proc. Eighth Workshop on Empirical Studies of Software Maintenance* (WESS'02), Montreal, CA, 2002.

M.M. Lehman and J.F. Ramil, Software Evolution, in this volume, 2004.

M.M. Lehman, J.F. Ramil, and G. Kahen, *A Paradigm for the Behavioural Modelling of Software Processes using System Dynamics*, technical report, Dept. of Comp., Imperial College, London, September 2001.

M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, and W. Turski, Metrics and Laws of Software Evolution – The Nineties View, *Proc. 4th Intern. Symp. Software Metrics*, 20-32, Albuquerque, NM, November 1997.

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

T. Mens, J. Buckley, M. Zenger, and A. Rashid, Towards a Taxonomy of Software Evolution, *Second Intern. Workshop on Unanticipated Software Evolution,* Warsaw, Poland, April 2003. http://joint.org/use2003/Papers/18500066.pdf

M.H. Meyer and J.M. Utterback, The Product Family and the Dynamics of Core Capability, *Sloan Management Review*, 34(3), 29-47, Spring 1993.

P. Mi and W. Scacchi, A Knowledge-Based Environment for Modeling and Simulating Software Engineering Processes, *IEEE Trans. Data and Knowledge Engineering*, 2(3), 283-294, September 1990. Reprinted in *Nikkei Artificial Intelligence*, 20(1), 176-191, January 1991 (in Japanese); also in *Process-Centered Software Engineering Environments*, P.K. Garg and M. Jazayeri (eds.), IEEE Computer Society, 119-130, 1996.

P. Mi and W. Scacchi, Process Integration in CASE Environments, *IEEE Software*, 9(2), 45-53, March 1992. Reprinted in Eliot Chikofsky (ed.), *Computer-Aided Software Engineering (CASE)*, Second Edition, IEEE Computer Society, 1993.

P. Mi and W. Scacchi, A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330, 1996.

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

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

B. Nardi and V. O'Day, *Information Ecologies: Using Technology with Heart*, MIT Press, Cambridge, MA 1999.

R.R. Nelson and S.G. Winter, *An Evolutionary Theory of Economic Change*, Belknap Press, Cambridge, MA, 1982.

M.H. Nitecki, (ed.), *Evolutionary Progress*, University of Chicago Press, Chicago, IL, 1988.

S. O'Mahony, Developing Community Software in a Commodity World, in M. Fisher and G. Downey (eds.), *Frontiers of capital: Ethnographic Reflections on the New Economy*, Social Science Research Council, (to appear), 2003.

J.W. Paulson, G. Succi, and A. Eberlein, An Empirical Study of Open-Source and Closed-Source Software Products, *IEEE Trans. Software Engineering*, 30(4), 246-256, April 2004.

D.E. Perry, and J.F. Ramil, Empirical Studies of Software Evolution, in this volume, 2004.

D.E. Perry, H.P. Siy, and L.G. Votta, Parallel Changes in Large-Scale Software Development: An Observational Case Study, *ACM Trans. Software Engineering and Methodology*, 10(3), 308-337, 2001.

K.R. Popper, *Conjectures and Refutations*, Routledge & Kagen, 1963.

C.R. Reis and R.P.M. Fortes, An Overview of the Software Engineering Process and Tools in the Mozilla Project, *Proc. Workshop on Open Source Software Development*, 155-175, Newcastle, UK, February 2002.

G. Robles-Martinez, J.M. Gonzalez-Barahona, J. Centeno Gonzalez, V. Matellan Olivera, and L. Rodero Merino, Studying the Evolution of Libre Software Projects using Publicly

Available Data, *Proc. 3rd Workshop on Open Source Software Engineering*, Portland, OR, 2003.

P.P. Saviotti and G.S. Mani, Competition, Variety and Technological Evolution: A Replicator Dynamics Model, *J. Evolutionary Economics*, 5(4), 369-92, 1995.

W. Scacchi, Understanding Software Productivity: Towards a Knowledge-Based Approach, *Intern. J. Software Engineering and Knowledge Engineering*, 1(3), 293-321, 1991. Revised version in D. Hurley (ed.), *Advances in Software Engineering and Knowledge Engineering*, Volume 4, 37-70, 1995.

W. Scacchi, Experiences in Software Process Simulation and Modeling, *J. Systems and Software*, 46(2/3), 183-192, 1999

W. Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings – Software*, 149(1), 24-39, February 2002a.

W. Scacchi, Process Models for Software Engineering, in J. Marciniak (ed.), *Encyclopedia of Software Engineering,* 2nd Edition, John Wiley and Sons Inc., New York 993-1005, 2002b.

W. Scacchi, *Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development*, technical report, Institute for Software Research, July 2002c.

W. Scacchi, Free/Open Source Software Development in the Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004.

W. Scacchi and P. Mi, Process Life Cycle Engineering: Approach and Support Environment, *Intern. J. Intelligent Systems in. Accounting, Finance, and Management*, 6:83-107, 1997.

S.R. Schach, B. Jin, D.R. Wright, G.Z. Heller, and A.J. Offutt, Maintainability of the Linux Kernel, *IEE Proceedings – Software*, 149(1), 18-23, February 2002.

N. Smith and J.F. Ramil, Qualitative Simulation of Software Evolution Processes, *WESS'02 Eighth Workshop on Empirical Studies of Software Maintenance*, Montreal, October 2002.

N. Smith, A. Capiluppi, and J.F. Ramil, Qualitative Analysis and Simulation of Open Source Software Evolution, *Proc. 5th Software Process Simulation and Modeling Workshop (ProSim'04),* Edinburgh, Scotland, UK, May 2004.

M. Svahnberg and J. Bosch, Evolution in Software Product Lines, *J. Software Maintenance*, 11(6), 391-422, 1999.

T. Tamai and Y. Torimitsu, Software Lifetime and its Evolution Process over Generations, *Proc. Conf. Software Maintenance*, 63-69, November 1992.

W. Turski, Reference Model for Smooth Growth of Software Systems, *IEEE Trans. Software Engineering*, 22(8), 599-600, August 1996.

J.M. Utterback, *Mastering the Dynamics of Innovation: How Companies Can Seize Opportunities in the Face of Technological Change*, Harvard Business School Press, 1994.

N.M. Victor and J.H. Ausubel, DRAMs as Model Organisms for Study of Technological Evolution, *Technological Forecasting and Social Change*, 69(3), 243-262, April 2002.

J. van den Ende and R. Kemp, Technological transformations in history: how the computer regime grew out of existing computing regimes, *Research Policy*, 28, 833-851, 1999.

E. von Hippel and R. Katz, Shifting Innovation to Users via Toolkits, *Management Science*, 48(7), 821-833, July 2002.

R. Yin, *Case Study Research: Design and Methods (Second Edition)*, Sage Publications, Newbury Park, CA. 1994.

Modeling OSSD Processes and Practices

This section contains the following five chapters. The first examines the modeling of a new class of hidden and knowledge-intensive processes associated with the recruitment and migration of people within a large OSSD project.

The second describes our collective experience across a number of years of studying different OSSD projects in discovering, modeling, and computationally re-enacting the processes we have found along the way.

The third provides a comprehensive examination of a multi-enterprise project network that spans and interconnects the processes of each project, as well as processes that can communicate and share knowledge-intensive work artifacts across project boundaries.

The third chapter demonstrates the key methodological result that appeared in the original proposal, namely to present a scheme that associates ethnographic analysis with computational approaches to model, analyze, and explain open source software development practices and processes. This is the first such work of its kind, and this chapter represents a copy of a paper that recognized as the *Best Paper* at the 1st Intern. Conference on Open Source Software, Genoa, IT, July 2005.

Chris Jensen and Walt Scacchi, Modeling Recruitment and Role Migration Processes in OSSD Projects, *Proc. 6th Intern. Workshop on Software Process Simulation and Modeling*, St. Louis, MO, May 2005.

Chris Jensen and Walt Scacchi, Process Modeling Across the Web Information Infrastructure, *Software Process--Improvement and Practice*, 10(3), 255-272, July-September 2005.

Chris Jensen and Walt Scacchi, Experiences in Discovering, Modeling, and Reenacting Open Source Software Development Processes, to appear in Mingshu Li, Barry Boehm, and Leon J. Osterweil (eds.), *Unifying the Software Process Spectrum: Proc. Software Process Workshop*, Beijing, China, May 2005, Springer-Verlag, 2005.

Walt Scacchi, Chris Jensen, John Noll, and Margaret Elliott, Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes, *Proc. First Intern. Conf. Open Source Software*, Genoa, Italy, July 2005.

Modeling Recruitment and Role Migration Processes in OSSD Projects

Chris Jensen and Walt Scacchi
Institute for Software Research
Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA USA 92697-3425
{cjensen, wscacchi}@ics.uci.edu

Abstract

Socio-technical processes have come to the forefront of recent analyses of the open source software development (OSSD) world. Though there many anecdotal accounts of these processes, such narratives lack the precision of more formal modeling techniques, which are needed if these processes are going to be systematically analyzed, simulated, or re-enacted. Interest in making these processes explicit is mounting, both from the commercial side of the industry, as well as among spectators who may become contributors to OSSD organization. Thus, the work we will discuss in this paper serves to close this gap by analyzing and modeling recruitment and role transition processes across three prominent OSSD communities whose software development processes we've previously examined: Mozilla.org, the Apache community, and NetBeans.

Keywords: Project recruitment, membership, process modeling, open source, Mozilla, Apache, NetBeans

Introduction

In recent years, organizations producing both open and closed software have sought to capitalize on the perceived benefits of open source software development methodologies. This necessitates examining the culture of prominent project communities in search of ways of motivating developers. Although the ensuing studies have provided much insight into OSSD culture, missing from this picture was the process context that produced the successes being observed. Ye and Kishida (2003) and Crowston and Howison (2005) observe that community members gravitate towards central roles over time represented with "onion" diagrams such as in figure 1. These depictions indicate a similar number of layers in organizational hierarchies across communities, but do not suggest how one might transition between layers and what roles are available at each layer. Much like their development processes, OSSD communities typically provide little insight into role migration processes. What guidance is provided is often directed at recruitment- initial steps to get people in the door. Guidance for attaining more central roles is often characterized as being meritocratic, depending on the governance structure of the community. Nevertheless, these development roles and how developers move between them seems to lie outside of the traditional view of software engineering, where developers seem to be limited to roles like requirements analyst, software designer, programmer, or code tester, and where there is little/no movement between roles (except perhaps in small projects).

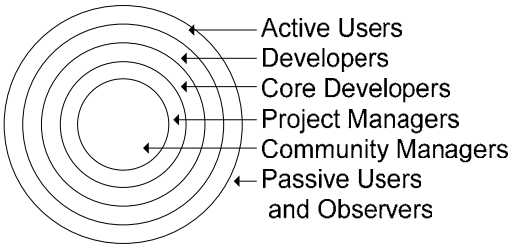


Figure 1. An "onion" diagram representation of an open source community organizational hierarchy

Christie and Staley (2000) argue that social and organizational processes, such as those associated with moving between different developer roles in a project, are important in determining the outcome of software development processes. In previous studies, we have examined software development processes within and across OSSD communities (Jensen and Scacchi, 2005, Scacchi 2002, 2004, 2005). Here, we take a look at two related socio-technical processes used in OSSD as a way of merging the social/cultural and technical/developmental OSSD activities. Specifically, we'll focus on the recruitment and migration of developers from end-users or infrequent contributors towards roles more central to the community, like core developer, within projects such as the Mozilla, Apache community, and NetBeans projects. Such processes characterize both the hierarchy of roles that OSS developers play (cf. Gacek and Arief 2004), as well as how developers move through or become upwardly mobile within an OSSD project (Sim and Holt 1998). While anecdotal evidence of these processes exists, the lack of precision in their description serves as a barrier to community entry,

continuous improvement, and process adoption by other organizations. The goal of our work here thus serves to provide process transparency through explicit modeling of such processes in ways that may enable increased community participation, more widespread process adoption, and process improvement.

In the remaining sections, we outline details about recruitment and role migration as membership processes as found while examining each of these three OSSD project communities. At the ProSim'05 Workshop we will present a variety of semi-structured and formal models that enable more rigorous analysis and simulated re-enactment using tools and techniques we have previously developed and employed (cf. Noll and Scacchi 2001, Jensen and Scacchi 2005)

Membership Processes in Mozilla.org

Developer recruitment in Mozilla was difficult at the start. The opening of the Netscape browser source code offered developers a unique opportunity to peek under the hood of the once most dominant Web browser in use. Nevertheless, the large scale of the application (millions of lines of source code) and the complex/convoluted architecture scared developers away. These factors, combined with the lack of a working release and the lack of support from Netscape led one project manager to quit early on (Mockus, *et. al*, 2002). However, with the eventual release of a working product, the Mozilla project garnered users who would become developers to further the cause.

The Mozilla Web site lists several ways for potential developers and non-technical people to get involved with the community (Getting Involved with Mozilla.org, 2005). The focus on quality assurance and documentation reflects a community focus on maturing, synchronizing, and stabilizing updates to the source code base. Technical membership roles and responsibilities currently listed include bug reporting, screening, confirming, and fixing, writing documentation, and contacting sites that do not display properly under Mozilla. Compared to more central roles, these activities do not require deep knowledge of the Mozilla source code or system architecture, and serve to allow would-be contributors to get involved and participate in the overall software development process.

When bugs are submitted to the Bugzilla, they are initially assigned to a default developer for correction. It is not uncommon for community developers and would-be developers to become frustrated with an outstanding issue within the bug repository and submit a patch, themselves.

The next task is to recruit others to accept the patch and incorporate it into the source tree. Recruitment of patch review is best achieved through emailing reviewers working on the module for which the patch was committed or reaching out to the community via the Mozilla IRC chat. By repeatedly demonstrating competency and dedication writing useful code within a section of the source, would-be developers gain a reputation among those with commit access to the current source code build tree. Eventually, these committers recommend that the developer be granted access by the project drivers. In rare cases, such a developer may even be offered ownership of a particular module if s/he is the primary developer of that module and it has not been blocked for inclusion into the trunk of the source tree[1].

Once a project contributor is approved as a source code contributor, there are several roles available to community members. Most of these are positions requiring greater seniority or record of demonstrated accomplishments within the community. As module developers and owners establish themselves as prominent community members, other opportunities may open up. In meritocratic fashion (cf. Fielding 1999), developers may transition from being a QA module contact to a QA owner. Similar occasions exist on the project level for becoming a module source reviewer.

Super-reviewers attain rank by demonstrating superior faculty for discerning quality and effect of a given section of source on the remainder of the source tree. If a reviewer believes that s/he has done this appropriately, s/he must convince an existing super-reviewer of such an accomplishment. This super-reviewer will propose the candidate to the remainder of the super-reviewers. Upon group consensus, the higher rank is bestowed on the reviewer (Mozilla Code Review FAQ, 2005). The same follows for Mozilla drivers, who determine the technical direction of the project per release.

Community level roles include smoke-test coordinator, code sheriff, and build engineer, although no process is prescribed for such transitions. As individual roles, they are held until vacated, at which time, the position is filled by appointment from the senior community members and Mozilla Foundation staff. Role hierarchy and a flow graph of the migration process for transitioning from reviewer to super-reviewer are provided in figure 2 as an example of those we have modeled for this community. In the flow graph, rectangles refer to actions, whereas ovals

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=18574

refer to resources created or consumed by the associated action, as determined by the direction of the arrow linking the two. Transitions from one role to another are depicted with a dashed arrow from an action performed by one role to the title of another. We have also used dashed lines to differentiate social or role transitioning activities and resources from strictly technical, developmental resources.

Membership Processes in the Apache Community

Role migration in the Apache community is linear. The Apache Software Foundation (ASF) has laid out a clear path for involvement in their meritocracy. Individuals start out as end-users (e.g., Web site administrators), proceed to developer status, then committer status, project management committee (PMC) status, ASF membership, and lastly, ASF board of directors membership (How the ASF Works, 2005). Much as in advancement in the Mozilla community, Apache membership is by invitation only. As the name suggests, the Apache server is comprised of patches submitted by developers. These patches are reviewed by committers and either accepted or rejected into the source tree.

In addition to feature patches, developers are also encouraged to submit defect reports, project documentation, and participate on the developer mailing lists. When the PMC committee is satisfied with the developer's contributions, they may elect to extend an offer of "committership" to the developer, granting him/her write access to the source tree. To accept committership, the developer must submit a contributor license agreement, granting the ASF license to the intellectual property conveyed in the committed software artifacts.

PMC membership is granted by the ASF. To become a PMC member, the developer/committer must be nominated by an existing ASF member and accepted by a majority vote of the ASF membership participating in the election (Fielding, et. al, 2002). Developers and committers nominated to become PMC members have demonstrated commitment to the project, good judgment in their contributions to the source tree, and capability in collaborating with other developers on the project. The PMC is responsible for the management of each project within the Apache community. The chair of the PMC is an ASF member elected by his/her fellow ASF members who initially organizes the day-to-day management infrastructure for each project, and is ultimately responsible for the project thereafter. ASF membership follows the same process as PMC membership- nomination and election by a majority vote of existing ASF members.

ASF members may run for office on the ASF board of directors, as outlined by the ASF bylaws (Bylaws of the Apache Software Foundation, 2005). Accordingly, the offices of chairman, vice chairman, president, vice president, treasurer (and assistant), and secretary (and assistant) are elected annually. A flow graph of the role migration process appears in figure 3.

Although, there is one path of advancement in the Apache community, there are several less formal committees that exist on a community (as opposed to project) scale. These include the conference organizing committee, the security committee, the public relations committee, the Java Community Process (JCP) committee, and the licensing committee. Participation in these committees is open to all committers (and higher ranked members) and roles are formalized on an as-needed basis (e.g. conference organization). Non-committers may apply for inclusion in specific discussion lists by sending an email to the board mailing alias explaining why access should be granted. Thus, processes associated with these committees are ad hoc and consist of one step.

Membership Processes in the NetBeans.org Community

Roles in the NetBeans.org community for developing the Java-based NetBeans interactive development environment are observable on five levels of project management (Oza, et. al 2002) just as in Apache. These range from users to source contributors, module-level managers, project-level managers, and community-level managers. The NetBeans community's core members are mostly Sun Microsystems employees, the community's primary sponsor, and are subject to the responsibilities set on them by their internal organizational hierarchy. As such, (and unlike the cases of Apache and Mozilla), not all roles are open to volunteer and third-party contributors. Non-Sun employed community members wanting to participate beyond end-usage are advised to start out with activities such as quality assurance (QA), internationalization, submitting patches, and documentation (Contributing to the NetBeans Project, 2005). As in the case with Mozilla, until they have proven themselves as responsible, useful, and dedicated contributors, developers must submit their contributions to developer mailing lists and the issue repository, relying on others with access to commit the source. However, unlike Mozilla, developers are also encouraged to start new modules.

While the community was more liberal with module creation early in the project's history, as the community has matured, additions to the module catalogue have become more managed to eliminate an abundance of abandoned modules. Also as in Mozilla, developers are subjected to the proving themselves before being granted committer status on a portion of the source tree. Additionally, they may gain module owner status be creating a module or taking over ownership of an abandoned module that they have been the primary committer for. With module ownership comes the responsibility to petition the CVS manager to grant commit access to the source tree to developers, thereby raising their role status to "committer."

Rising up to the project-level roles, the Sun-appointed CVS source code repository manager is responsible for maintaining the integrity of the source tree, as well as granting and removing developer access permissions. In contrast, the release manger's role is to coordinate efforts of module owners to plan and achieve timely release of the software system. Theoretically, any community member may step in at any time and attempt to organize a release. In practice, this rarely occurs. Instead, most community members passively accept the roadmap devised by Sun's NetBeans team. In the latter case, the previous release manager puts out a call to the community to solicit volunteers for the position for the upcoming cycle. Assuming there are no objections, the (usually veteran) community member's candidacy is accepted and the CVS manager prepares the source tree and provides the new release manager permissions accordingly. Alternatively, a member of Sun may appoint a member of their development team to head up the release of their next development milestone.

At the community-management level, the community managers coordinate efforts between developers and ensures that issues brought up on mailing lists are addressed fairly. At the inception of the NetBeans project, an employee of CollabNet (the company hosting the NetBeans Web portal) originally acted as community manager and liaison between CollabNet and NetBeans. However, it was soon transferred to a carefully selected Sun employee (by Sun) who has held it since. As community members have risen to more central positions in the NetBeans community, they tend to act similarly, facilitating and mediating mailing list discussions of a technical nature, as well as initiating and participating in discussions of project and community direction.

Lastly, a committee of three community members, whose largely untested responsibility is to ensure fairness within the community, governs the NetBeans project.. One of the three is appointed by Sun. The community at large elects the other two members of the governance board. These elections are held every six months, beginning with a call for nominations by the community management. Those nominees that accept their nomination are compiled into a final list of candidates to be voted on by the community. A model of the product development track role migration process is shown in figure 4.

Discussion

In both NetBeans and Mozilla, recruitment consists of listing ways for users and observers to get involved. Such activities include submitting defect reports, test cases, source code and so forth. These activities require a low degree of interaction with other community members, most notably decision makers at the top of the organizational hierarchy. Our observation has been that the impact of contributions trickles up the organizational hierarchy whereas socio-technical direction decisions are passed down. As such, activities that demonstrate capability in a current role, while also coordinating information between upstream and downstream (with respect to the organizational hierarchy) from a given developer are likely to demonstrate community member capability at his/her current role, and therefore good candidates for additional responsibilities.

Recruitment and role migration processes aren't something new; since they describe the actions and transition passages involved in moving along career paths. Like career paths described in management literature (e.g., Lash and Sein 1995), movement in the organizational structure may be horizontal or vertical. Most large OSSD project communities are hierarchical, even if here are few layers to the hierarchy and many members exist at each layer.

In the communities we have examined, we found different paths (or tracks) towards the center of the developer role hierarchy as per the focus of each path. Paths we've identified include project management (authority over technical issues) and organizational management (authority over social/infrastructural issues). Within these paths, we see tracks that reflect the different foci in their software processes. These include quality assurance roles, source code creation roles, and source code versioning roles (e.g. cvs manager, cvs committer, etc), as well as role paths for usability, marketing, and licensing. There are roles for upstream development activities (project planning--these are generally taken up by more senior members of the community. This is due in part that developers working in these roles can have an

impact on the system development commensurate with the consequences/costs of failure, and require demonstrated skills to ensure the agents responsible won't put the software source code into a state of disarray).

In comparison to traditional software development organizations, tracks of advancement in open source communities are much more fluid. A developer contributing primarily to source code generation may easily contribute usability or quality assurance test cases and results to their respective community teams. This is not to suggest that a module manager of a branch of source code will automatically and immediately gain core developer privileges, responsibilities, and respect from those teams. However, industrial environments tend towards rigid and static organizational hierarchies with highly controlled growth at each layer.

The depiction of role hierarchies in open source communities as concentric, onion-like circles speaks to the fact that those in the outer periphery have less direct control or knowledge of the community's current state and its social and technical direction compared to those in the inner core circle. Unlike their industrial counterparts, open source community hierarchies are dynamic. Although changes in the number of layers stabilizes early in the community formation, the size of each layer (especially the outer layers) is highly variable. Evolution of the organizational structure may cause or be caused by changes in leadership, control, conflict negotiation, and collaboration in the community, such as those examined elsewhere (Jensen and Scacchi 2005b). If too pronounced, these changes can lead to breakdowns of the technical processes.

As a general principle, meritocratic role migration processes such as those we have observed consist of a sequence of establishing a record of contribution in technical processes in collaboration with other community members, followed by certain "rights of passage" specific to each community. For Apache, there is a formal voting process that precedes advancement. However, in the Mozilla and NetBeans communities, these are less formal. The candidate petitions the appropriate authorities for advancement or otherwise volunteers to accept responsibility for an activity. These authorities will either accept or deny the inquiry.

Conclusion

Social or organizational processes that affect or constrain the performance of software development processes have had comparatively little investigation. This is partially because some of these processes may be well understood (e.g., project management processes like scheduling or staffing), while others are often treated as "one-off" or *ad hoc* in nature, executing in a variety of ways in each instantiation. The purpose of our examination and modeling study of recruitment and role migration processes is to help reveal how these socio-technical processes are intertwined with conventional software development processes, and thus constrain or enable how software processes are performed in practice. In particular, we have examined and modeled these processes within a sample of three OSSD projects that embed the Web information infrastructure. Lastly, we have shown where and how they interact with existing software development processes found in our project sample.

References

Bylaws of the Apache Software Foundation, available online at http://www.apache.org/foundation/bylaws.html accessed 7 February 2005

Christie, A. and Staley, M. "Organizational and Social Simulation of a Software Requirements Development Process" *Software Process Improvement and Practice* 2000; 5: 103–110 (2000)

Contributing to the NetBeans Project, available online at http://www.netbeans.org/community/contribute/ accessed 7 February 2005

Coward, Anonymous. "About Firefox and Mozilla" Comment on Slashdot.org forum "Firefox Developer on Recruitment Policy," available online at http://developers.slashdot.org/comments.pl?sid=137815&threshold=1&commentsort=0&tid=154&tid=8&mode=thread&cid=11527647, 31 January, 2005.

Crowston, K. and Howison, J. 2005. The Social Structure of Free and Open Source Software Development, First Monday, 10(2). February. Online at http://firstmonday.org/i8ssues/issue10_2/crowston/index.html

Elliott, M., The Virtual Organizational Culture of a Free Software Development Community, *Proceedings of the Third Workshop on Open Source Software*, Portland, Oregon, May 2003.

Fielding, R., Shared Leadership in the Apache Project. *Communications ACM*, 42(4), 42-43, 1999.

Fielding, R., Hann, I-H., Roberts, J and Sandra Slaughter, S. "Delayed Returns to Open Source Participation: An Empirical Analysis of the Apache HTTP Server Project," Presented at the Conference on Open Source: Economics, Law, and Policy, Toulouse, France June 2002.

Gacek, C. and Arief, B., The Many Meanings of Open Source, *IEEE Software*, 21(1), 34-40, January/February 2004.

Getting Involved with Mozilla.org, Web page available online at http://www.mozilla.org/contribute/ 3 November 2004

How the ASF works, available online at http://www.apache.org/foundation/how-it-works.html, accessed 7 February 2005

Jensen, C. and Scacchi, W., Process Modeling Across the Web Information Infrastructure, *Software Process Improvement and Practice*, to appear, 2005.

Jensen, C. and Scacchi, W. Collaboration, Leadership, Control, and Conflict Negotiation Processes in the NetBeans.org Open Source Software Development Community. working paper, Institute for Software Research, March 2005

Lash, P.B. and Sein, M.K. Career Paths in a Changing IS Environment: A Theoretical Perspective, Proc. SIGCPR 1995, 117-130. Nashville, TN

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

Mozilla Code Review FAQ, available online at http://www.mozilla.org/hacking/code-review-faq.html, accessed 7 February 2005

Noll, J. and Scacchi, W., Specifying Process-Oriented Hypertext for Organizational Computing, *J. Network and Computer Applications*, 24(1), 39-61, 2001.

Oza, M. Nistor, E. Hu, X., Jensen, C., Scacchi, W. "A First Look at the NetBeans Requirements and Release Process." June 2002, updated February 2004 available online at http://www.isr.uci.edu/~cjensen/papers/FirstLookNetBeans/.

Scacchi, W., Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39, February 2002.

Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004.

Scacchi, W., 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, 2005.

Sim, S.E. and Holt, R.C., 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, 1998.

Ye, Y. and Kishida, K. Towards an Understanding of the Motivation of Open Source Software Developers, *Proc. 25th Intern. Conf. Software Engineering*, Portland, OR, 419-429, 2003.

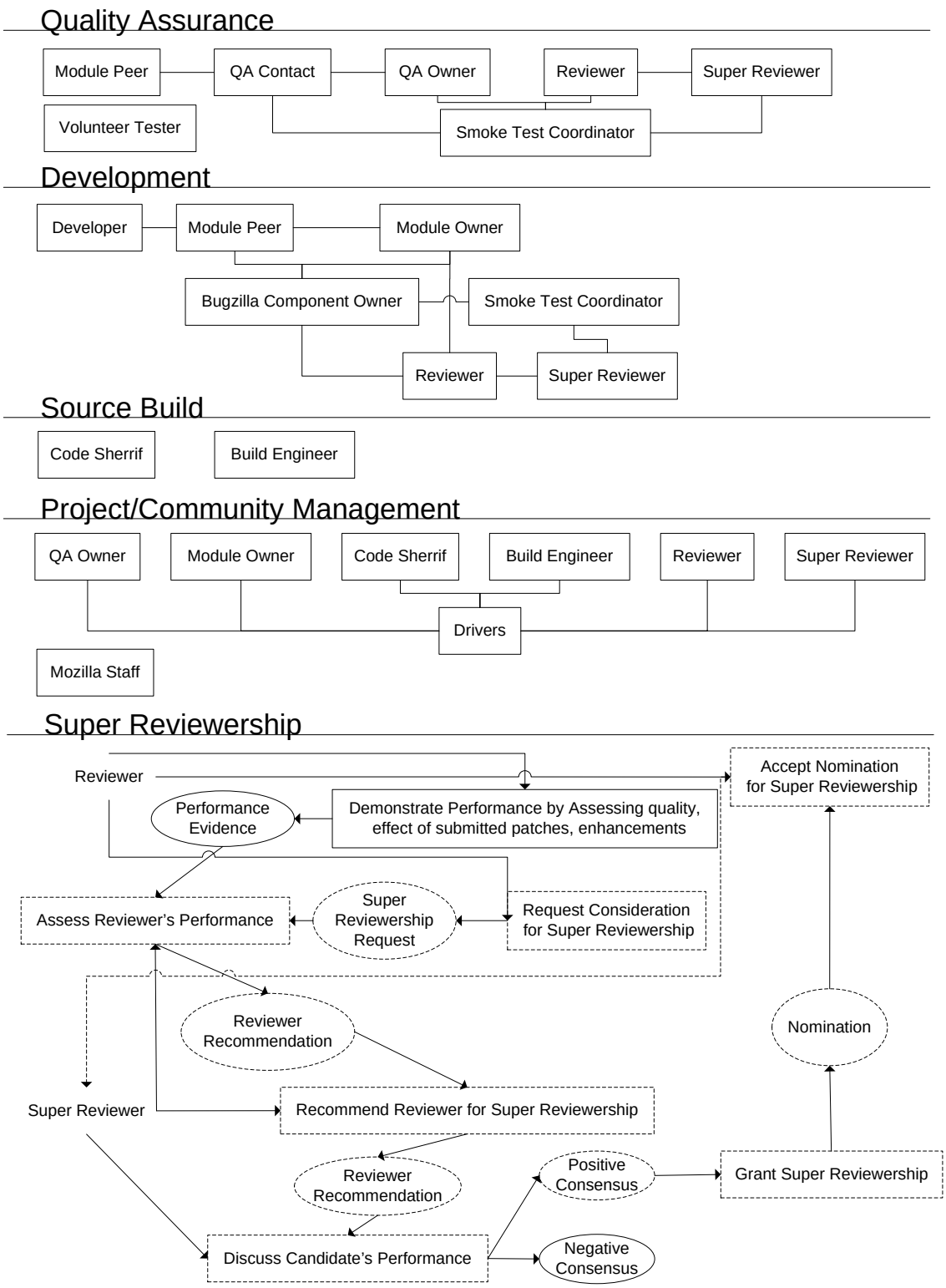


Figure 2. Role hierarchy and super reviewership migration in the Mozilla community

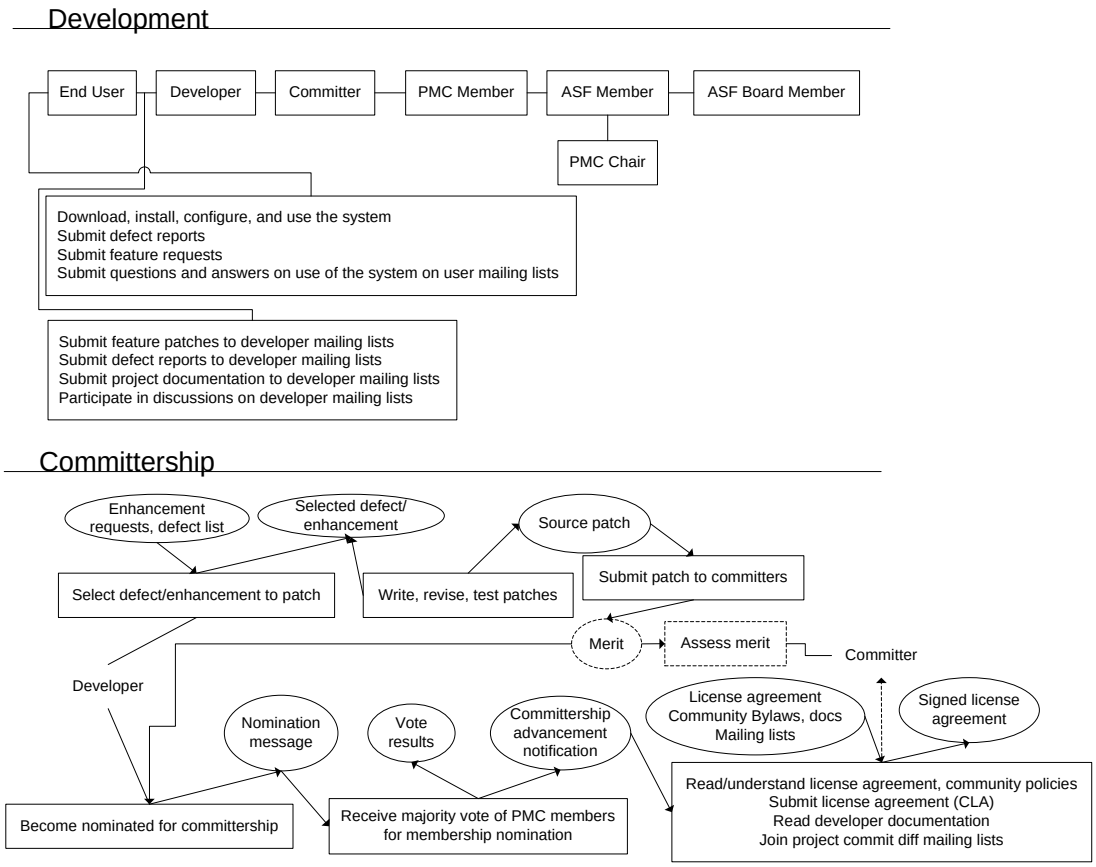


Figure 3. Role hierarchy and committership migration in the Apache community, highlighting the sequence of a developer becoming a committer

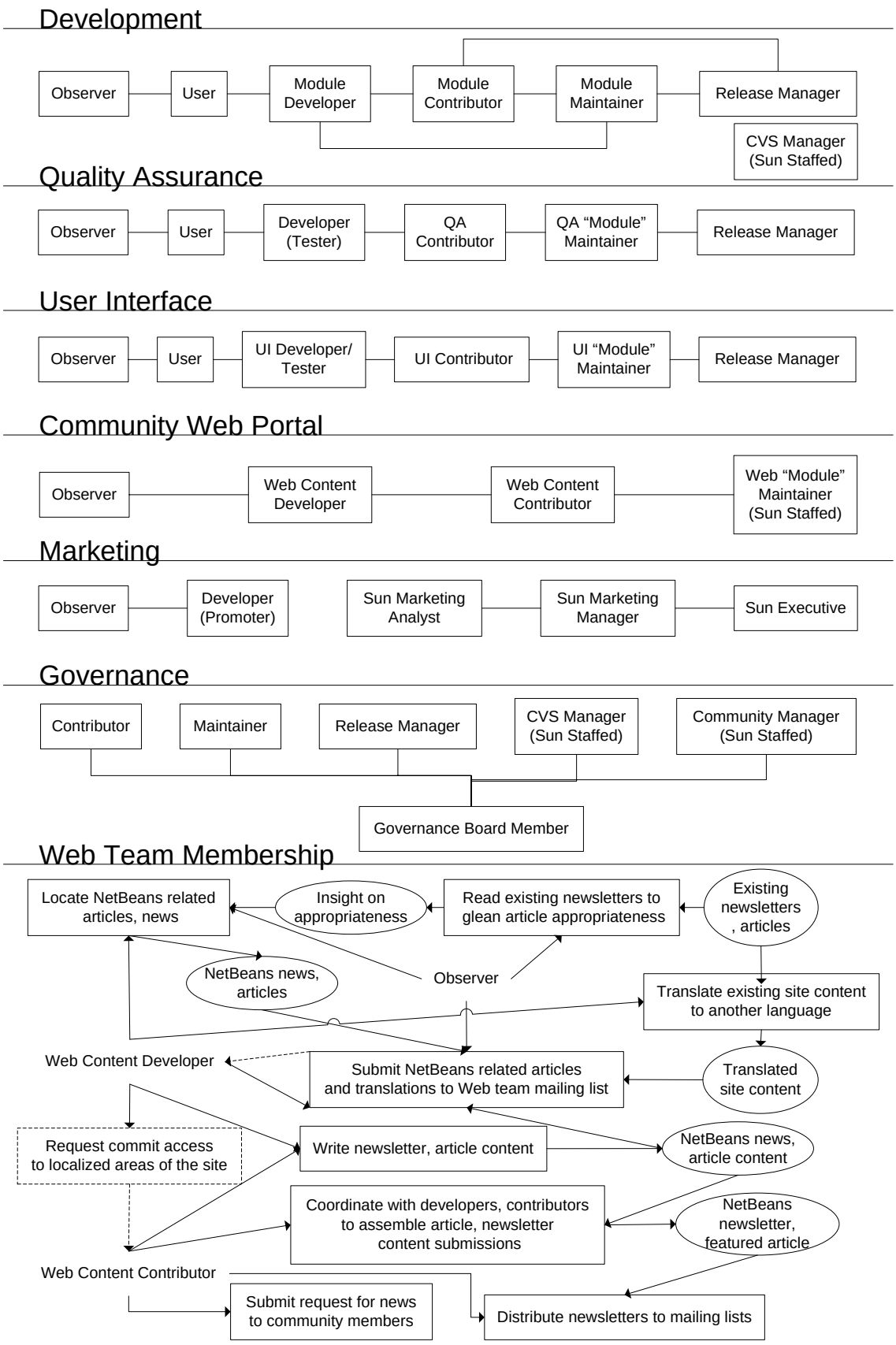


Figure 4. Role hierarchy and Web Team membership migration in the NetBeans open source community

SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2005; **10**: 255–272
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spip.228

Process Modeling Across the Web Information Infrastructure

Research Section

Chris Jensen*,† and Walt Scacchi
Institute for Software Research, University of California-Irvine, Irvine, CA, USA

Web-based open source software development (OSSD) project communities provide interesting and unique opportunities for software process modeling and simulation. While most studies focus on analyzing processes in a single organization, we focus on modeling software development processes both within and across three distinct but related OSSD project communities: Mozilla, a Web artifact consumer; the Apache HTTP server that handles the transactions of Web artifacts to consumers such as the Mozilla browser; and NetBeans, a Java-based integrated development environment (IDE) for creating Web artifacts and application systems. In this article, we look at the process relationships within and between these communities as components of a Web information infrastructure. We employ expressive and comparative techniques for modeling such processes that facilitate and enhance understanding of the software development techniques utilized by their respective communities and the collective infrastructure in creating them. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: interorganizational process modeling; process collaboration; process conflict; interorganizational interaction; process integration; open source software development; Apache; Mozilla; NetBeans

1. INTRODUCTION

Previous studies of interorganizational processes focus on devising languages to represent workflows across organizational boundaries (Rasch and Hansen 1997, Lenz and Oberweis 2001, Shen and Liu 2001) and verification of workflow nets (van der Aalst 2002b). Little work (van der Aalst 2002a) demonstrates real-life interorganizational processes. In contrast, this article contributes a framework for discovering, analyzing, and modeling interorganizational software processes within a multiproject software ecosystem. The ecosystem selected is a network of organizations responsible for core of the Web information infrastructure. Although nonopen source organizations are also members of this ecosystem, our focus is on open source software development (OSSD) organizations. Large-scale geographically distributed software development projects, such as OSSD project communities, present challenging process problems. The Apache, Mozilla, and NetBeans[1] OSSD communities collectively have millions of estimated users, and tens of thousands of community participants contributing in one fashion or another. Such magnitudes would be difficult for most closed

* Correspondence to: Chris Jensen, Institute for Software Research, University of California-Irvine, Irvine, CA, USA 92697-3425
† E-mail: cjensen@ics.uci.edu

Contract/grant sponsor: US National Science Foundation; contract/grant number: 0083075; 0205679; 0205724; 0350754

[1] The Eclipse project affiliated with IBM is similar in many ways to the NetBeans project, in that both efforts are very large OSS projects developing Java-based IDEs. But for our study, we selected the NetBeans project.

source organizations to manage. Yet, these three communities have proven extremely successful at it. Further, they have done so in a delicate ecosystem that includes evolving Web standards, data and software repositories, and tools. These serve as a framework for integrating each community's tools together. Therefore, as the components of this framework coevolve, each community must synchronize, (re)integrate, and stabilize its position within the process space.

In this article, we look at processes within and across three related OSSD project communities (cf. Scacchi 2002). In our efforts to model software development processes on both community and infrastructure levels, we have used a variety of techniques. These include detailed narrative models of processes, semistructured hyperlinked models, formal computational process models, and a reenactment simulator (Scacchi *et al*. 2004), all of which serve as input for other process engineering activities (Scacchi and Mi 1997, Noll and Scacchi 2001). Further, all of our process models are hypermedia artifacts that, when submitted back to the communities, may be consumed by the processes they describe.

From here, we will set the stage for our investigation with a discussion of each process modeled independently before examining the infrastructure as a whole in order to examine intercommunity process modeling issues and outcomes as we found them. Finally, we look at the modeling techniques themselves, how they may be used to guide developers, and how they can serve as a basis for process simulation and other process activities.

2. MODELING PROCESSES WITHIN WEB INFORMATION INFRASTRUCTURE PROJECTS

The Apache Web server, Mozilla Web browser, and NetBeans integrated development environment (IDE) together form a Web information infrastructure for developing and deploying Web-based software applications, content, and services (Fielding *et al*. 1998, Mockus *et al*. 2002). However, as the projects that develop each of these three open source software systems operate as virtual enterprises (Noll and Scacchi 1999), we have no basis to assume that their development process activities, roles, or tools are identical or common, nor how or where they interact. Thus, in order for these projects, and other OSSD projects like them, to collectively produce and sustain a viable global Web information infrastructure, they must be able at some point to synchronize and stabilize their processes, their process activities, shared artifacts, and targeted software releases (cf. Cusumano and Yoffie 1999).

Before we can understand software development processes across each of these three Web information infrastructure components, we must understand them individually. As previously introduced (Jensen and Scacchi 2003b, Scacchi *et al*. 2004), we address three process modeling techniques here as a sampling of those we have applied in our study. These are the rich hypermedia, process flow graphs, and formal modeling. Formal modeling in turn supports tools for simulated reenactment of software processes, which is used to preview, interactively walkthrough, validate (Atkinson and Noll 2003), and support process training on demand (Scacchi and Mi 1997). Additional details on these techniques can be found elsewhere (Scacchi *et al*. 2004).

This section seeks to address both of these issues. We start by presenting a brief overview of the *quality assurance (QA) process* in the Mozilla Web browser release cycle, modeling it as a rich hypermedia, followed by the Apache *release process,* modeled as a flow graph, and lastly, the NetBeans *requirements and release* process, which we model formally and reenact. For brevity, we will skip presenting the rich hypermedia, flow graphs, and formal models for each process; however, these models are detailed elsewhere (Ata *et al*. 2002, Carder *et al*. 2002, Jensen and Scacchi 2003b, Oza *et al*. 2002).

2.1. Rich Hypermedia Modeling with the Mozilla Quality Assurance Process

Building on and extending the rich picture concept described by Monk and Howard (1998), we created a rich hypermedia variant as an informal model of software development processes in each of Mozilla, Apache, and NetBeans projects. These models show the relationships between tools, agents, their development concerns (i.e. nonfunctional requirements), and activities that compose the overall process, and its functional requirements. While Monk and Howard propose a flat, static model, our hypermedia is interactive, deep, and navigational (Noll and Scacchi 2001), including process enactment

scenarios described and hyperlinked as use cases. Use cases are a known technique compatible with the Unified Modeling Language (UML) for representing user-system process enactment scenarios (Fowler 2000). The hypermedia artifacts are also annotated with detailed descriptions of each tool, agent, and concern. Each of these process objects is hyperlinked to its description. Descriptions can, in turn, be linked to other data or hypermedia resources. In this way, the modeler can define the scope of the rich hypermedia to include as little or as much information as the need requires. The rich hypermedia provides a quickly discernable intuition of the process without the burden of formalization. A rich hypermedia model for the Mozilla quality assurance process is shown as an image map in Figure 1.

The daily Mozilla QA cycle (Carder *et al*. 2002) begins with the closing of the source tree to submissions. After this, the 'code sheriff' and system build engineer create a build of the source code tree using the Mozilla Tinderbox build tool. If build errors are present, the sheriff and build engineer contact the 'on the hook' developers, reviewers, and super reviewers responsible for the build source, then they are called on via e-mail notifications to correct the defects. When the defect is corrected or the problematic source code is removed, the source is rebuilt. This process iterates until all build errors are corrected, and the compiled and built source and executable image is ready for the next process step.

The build results (including the executable software image) are placed on the community FTP server and the 'smoke test' coordinator issues a call for developers and volunteer testers via the community Internet relay chat (IRC) channel to download and evaluate the build results (e.g. execute the image on a test platform). After this, developers participating as QA contacts, QA owners, and volunteer

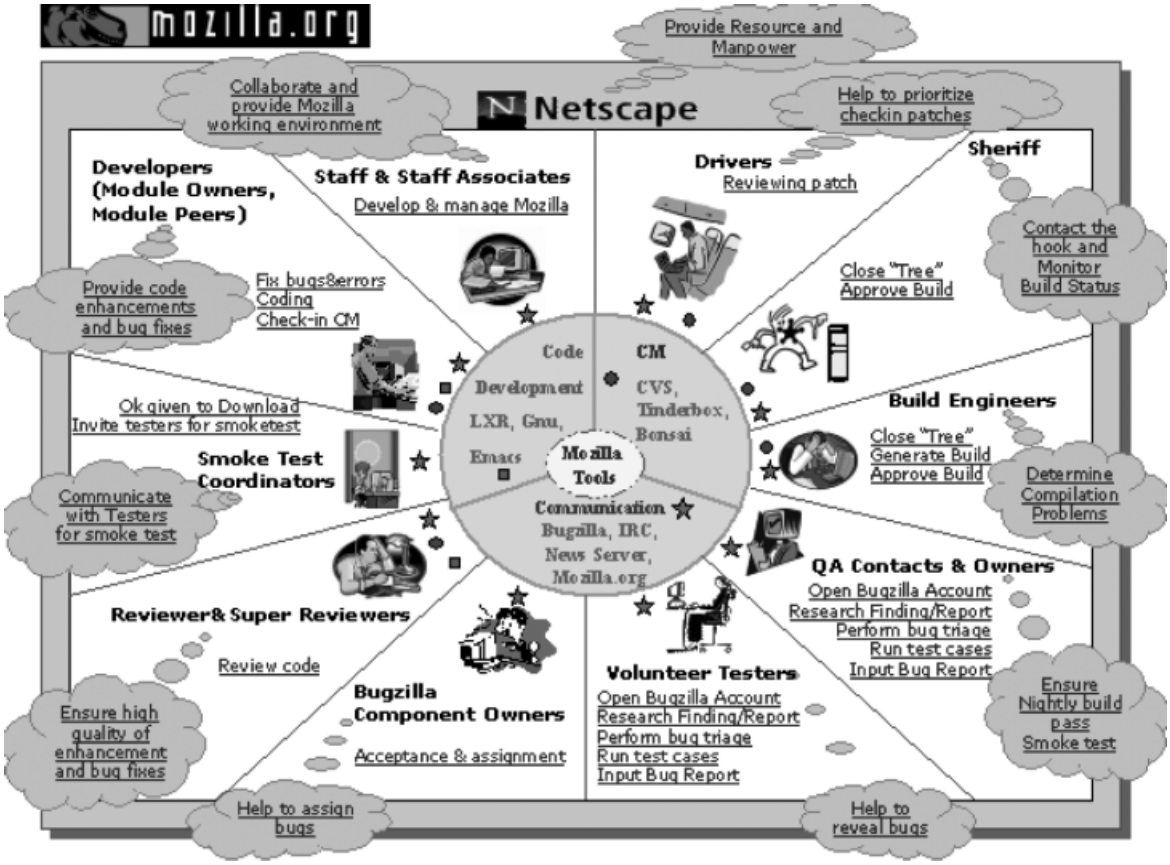


Figure 1. Mozilla quality assurance process rich hypermedia (cf. Carder *et al*. 2002)

testers will announce what they plan to test, download, and install the build and perform a series of smoke tests, security-specific (SSL) smoke tests, or less critical 'general tests' (periodic regression checkups), based on bug reports submitted to the bug repository. Testers note and discuss the results over the IRC channel. Critical bugs are identified and assigned to the on-the-hook developers to be patched, whereupon the source is retested. Noncritical bugs are set aside until another tester confirms them, uploaded to the Bugzilla defect repository, and further dealt with at a later time. Once all critical defects are corrected, the sheriff and build engineer reopen the source tree to further development and source submission.

When first detected, defects are entered into Bugzilla as unconfirmed, noting their severity, component, and platform where the defect was observed. A member of the quality assurance team (either a QA contact or owner) must then research the defect and certify it as a new defect or marking it as a duplicate of another known defect. Patches are then created by developers during the course of development or by drivers as the release date approaches to ensure the overall quality of the product, and the status revised to reflect the changes.

2.2. Process Flow Graph Modeling with the Apache HTTP Server Release Process

The process flow graph illustrates the flow of resources (development artifacts) through a path of interaction. The interaction accounts for process agents using tools that manipulate the resources through performance of tool-based activities. This semistructured workflow representation provides a partial ordering of the process fragments and allows us to tease out dependencies between artifacts and activities seen in the rich hypermedia. It also offers an idea of which artifacts and activities are most vital to development, by measuring the fan-in and fan-out of each.

These artifacts are likely to be the cause of bottlenecks in the development process when they are found to be inadequate, incomplete, or faulty results of prior development activities. Borrowing from Web modeling terminology, an artifact that is a hub or nexus for several activities will hold up development until it is completed or found satisfactory. Likewise, an artifact that is a product of several inputs inhibits activities that require it until it is ready for further processing. Additionally, we can also detect cycles of development (re)work, such as in the stabilization process and refining the software build release plan.

While these insights can be captured in other representational forms, this diagram, like the rich hypermedia, provides an overall representation of the context for process activities without the weight of the details of more formal models. Process entities shown in the flow graph may also be hyperlinked to resources in the community Web to provide interactive richness, as well as to enable process inspection activities. Figure 2 shows a process flow graph for the Apache HTTPD Server project's release process, where the boxes denote process activities and ellipses denote the resources or artifacts flowing through the process. Further, software developer roles are associated with each process activity.

As shown by this flow graph, in the Apache release process (Ata *et al*. 2002, Erenkrantz 2003), anyone can submit features or bug fixes in the form of patches to the server project. To be included in the project source repository, a 'committer' who has write access to the repository must upload the code. These patches are uploaded to the development branch of the repository. It may then be promoted to the stable release branch by a vote of the committers. While any developer can vote on an item, only the votes of primary author, committers, and members of the project management committee are binding (Fielding 1999). To be promoted, there must be at least three binding positive votes, and no vetoes. Further, for a release, there must be a majority approval (that is, at least three binding positive votes, and more positive than negative votes). Granting a developer committer status requires a complete consensus of the project management committee.

When a committer decides to create a release, she/he typically sends an announcement to fellow developers outlining the release plan, and stating who the release manager will be. Acceptance of the release plan requires a 'lazy consensus' approval of the committers[2]. As development moves towards completion, the release manager determines which features are fit for inclusion in the release and

[2] http://httpd.apache.org/dev/guidelines.html

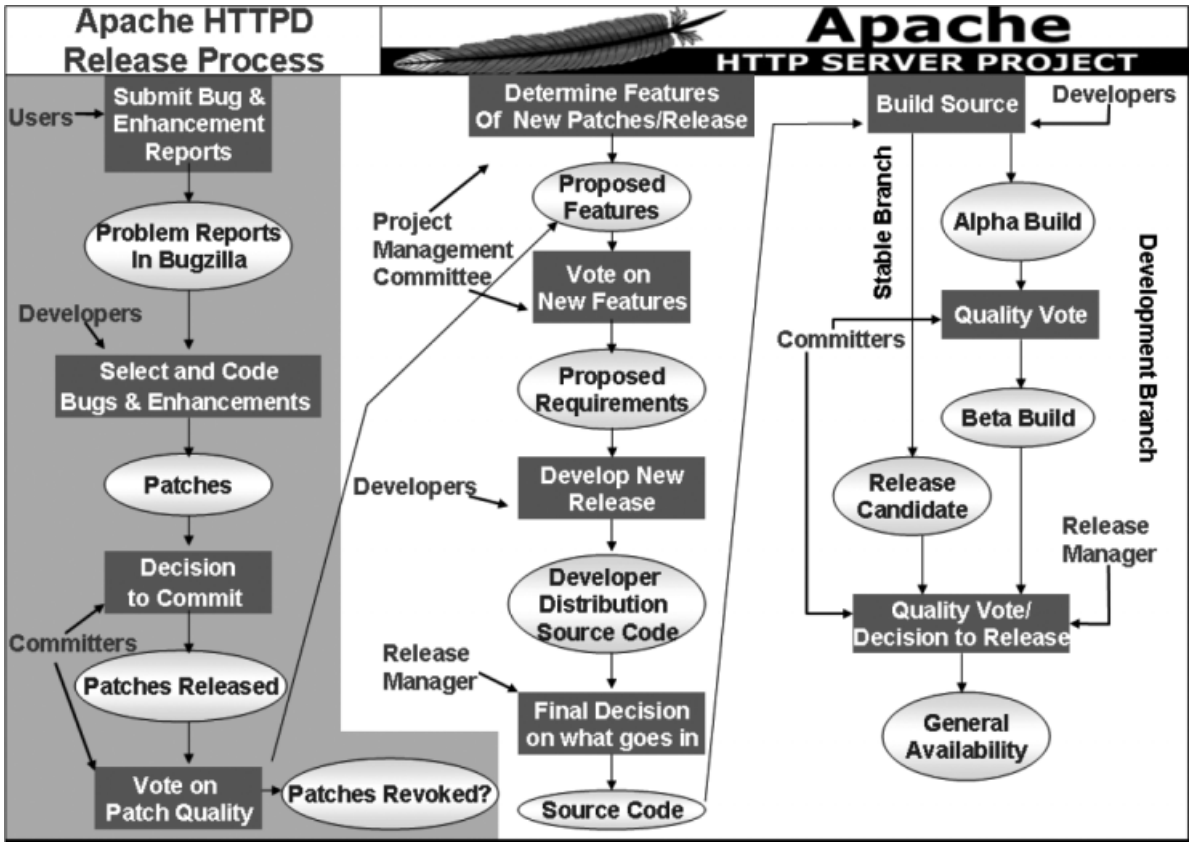


Figure 2. Apache HTTP server release process flow graph (cf. Ata *et al*. 2002)

which are not. Those that pass are compiled into an alpha build, which is made available on the community Web site and announced on the developer mailing lists. Developers and committers are then called upon to test the build on their own servers manually or through use of the automated Apache server test suite. Discovered defects are sent to the community development e-mail list (and potentially submitted to Bugzilla) and patched by developers and subsequently subjected to the patch review process. In the stable source branch, releases that pass the committer vote usually skip the beta and release candidate phase and proceed to general availability.

When the release manager is adequately satisfied with quality of the source in the development branch, she/he will declare the release suitable for beta or final release candidacy. When she/he announces this, the builds are made available on the main page of the community Web and adopted by a wider audience, for continued testing and patching. At some point, the release manager deems the source fit for general public use and creates a general availability build release, announcing it on the development, committer, and tester mailing lists. This build is then voted on by the committers and tested on the Apache community Web site. If there is a simple majority of approval and at least three positive votes, the release is declared final. The result is announced via the community Web and mailing lists, and then released for distribution via a system of mirrored Web sites.

2.3. Formal Modeling and Reenactment Simulation with the NetBeans Requirements and Release Process

We developed formal models of software processes following our preexisting process meta-model (Mi and Scacchi 1996) using Protege-2000 knowledge editing environment (Georgas 2002, Noy *et al*. 2001). The resulting models have the form of a semantic

web/hypertext (Noll and Scacchi 2001). The work done here is identifying instances for all the process meta-model components: agents, resources, tools, actions, and activity control flows, which we represent using the Protege-2000 tool. Once a process instance is input, it may be exported to an XML format, a graphical representation using the OntoViz tool, or to our process modeling language, PML (Noll and Scacchi 2001).

Protege-2000's editing and visualization facilities provide for a multitude of alternative views and visual rendering of the modeled process components, as well as their interrelationships and dependencies. As a result, the graphical rendering of a process model or process object-relation class views can at times be more intuitive than a coded textual format. Figure 3 shows a graphic representation of an underlying PML model of the NetBeans Requirements and Release process that has been interpreted for visual rendering and layout of its relational interdependencies. However, the textual PML representation can be used as input to other process engineering tools such as those supporting process enactment or process improvement.

The first step in the NetBeans requirements and release process (Oza *et al*. 2002, Jensen and Scacchi 2003b) is to establish a release manager, a set of development milestones (with estimated completion dates), and a central theme for the release. The theme is selected by the community members who have taken charge of the release, with the goal of overcoming serious deficiencies in the

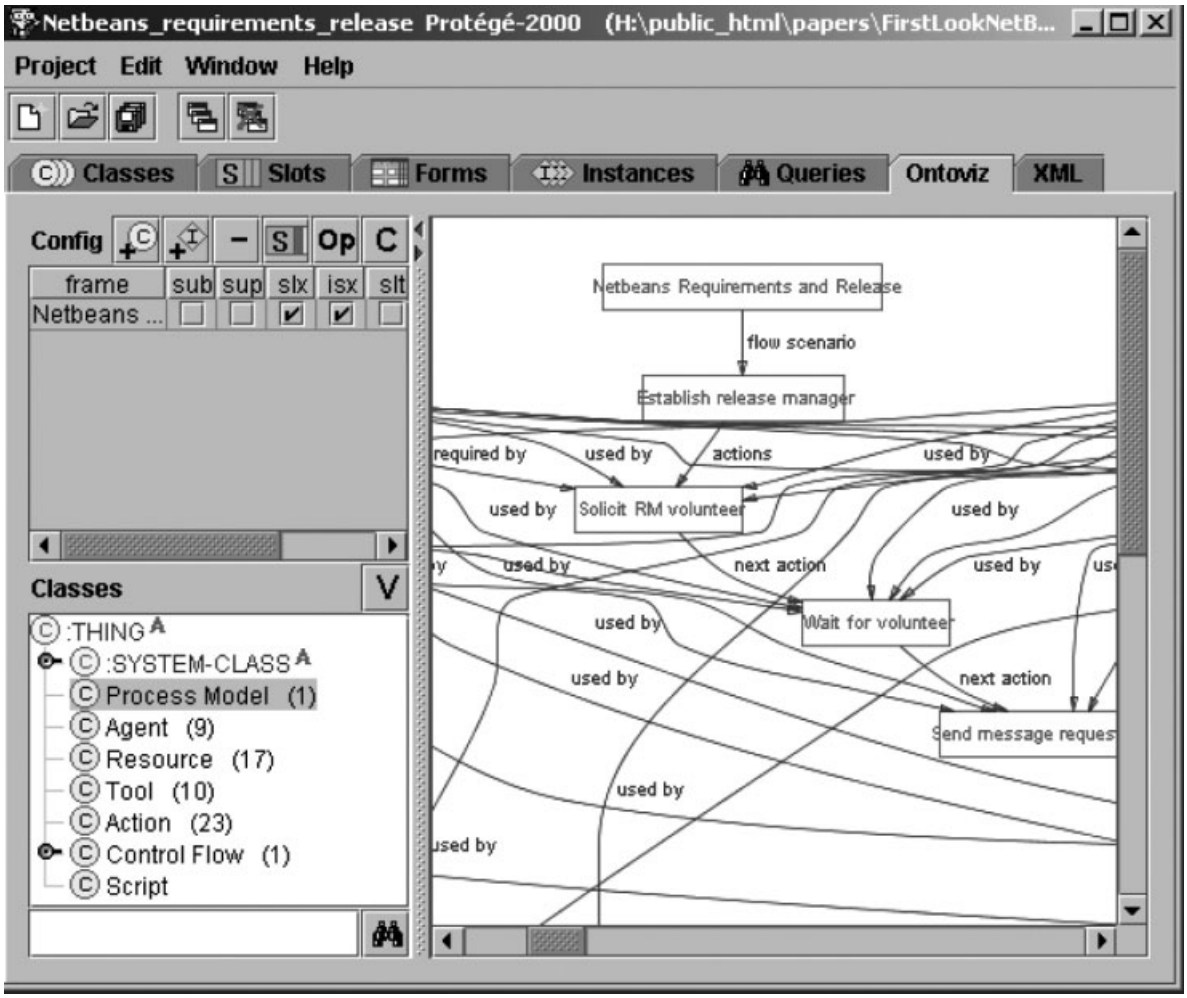


Figure 3. Visual rendering of the NetBeans requirements and release process formal model using Protégé-2000

product (e.g. quality, performance, and usability), in addition to new features and corrective maintenance planned by module teams. Historically, most releases have been led by members employed by Sun Microsystems, which provides development and financial support for the community, though volunteer releases also occur. On the basis of this, and in conjunction with input from the feature request reports, lead developers will draft a release plan, providing the milestones, target dates, and features to be implemented in the upcoming release. After review and revision by the community, the plan is accepted and developers are asked to volunteer to complete the tasks outlined therein and a volunteer is sought to act as release manager and coordinate efforts of community. Usually, a developer will either volunteer or be volunteered for the role via the mailing list by and accepts the nomination or is accepted through community consensus.

All creative development must be completed by the feature freeze milestone date specified in the release proposal, which signals the end of the requirements subprocess and the beginning of the stabilization phase, the release subprocess. At this point, only bug fixes may be submitted to the source tree. The stabilization phase consists of a build-test-debug cycle. Nightly builds are generated by a series of automated build scripts and subsequently subjected to a series of automated test scripts, the results of which are posted to the community Web site. Additionally, the quality assurance team performs a series of automated and manual testing every few weeks, as part of the Q-Build program with the aim of ensuring that source code submitted regularly meets reasonable quality standards. Defects discovered during testing are then recorded in the IssueZilla issue repository and subsequently corrected. When the release branch is believed to be devoid of critical 'show-stopping' defects, it is labeled a release candidate. If a week passes without any further showstoppers, the release candidate is declared final; else the defect is corrected and another release candidate is put forth. In addition to the formal depiction given in Figure 3, the NetBeans requirements and release process has also been reenacted. The motivation for this is as follows.

Process analysis seeks to identify potential pitfalls that can be discovered before or after their deployment or adoption in a project. Process model verifiers check static semantic properties such as whether a resource required downstream is provided by some upstream process activity (Atkinson and Noll 2003). Process simulators provide dynamic analysis capabilities through enactment or reenactment of processes, which are especially useful when validating, modifying, or redesigning a process, as well as for providing on-demand training (Scacchi and Mi 1997, Scacchi 2000, Atkinson and Noll 2003).

Our process enactment simulator (Choi and Scacchi 2001, Noll and Scacchi 2001) interactively serves a series of Web pages, or links to multiple alternative pages, according to the control flow expressed in the PML model of the process flow graph. This simulator allows process performers and other community members to simulate enacting the process through a step-by-step interactive walkthrough. With such a reenactment simulator, developers within a project may be able to exercise, critique, and identify improvement opportunities within processes that can be observed at a distance. It also provides the potential for a more easy transition from the simulator to live process enactment transactions on the community Web site.[3] In doing so, we have been able to detect processes whose workflows may be 'hidden', unseen, or unfamiliar to project participants with little/no involvement with the process. Similar analyses can also detect process flow segments that are unduly lengthy or otherwise suboptimal, which may serve as good candidates for process improvement or redesign. It allows for viewing and detection of the effects of duplicated work by participants that may be unaware of such duplication. Figure 4 thus displays a screen shot of one step during the reenactment of the NetBeans requirements and release process (Jensen and Scacchi 2003b). This display shows the process decomposition on the left, the enactment step, and corresponding link traversal options next, and then the enacted step's invocation on the right.

With the above insight into the development processes executing within Mozilla, Apache, and NetBeans, we can now explore development processes between them.

[3] For example, the NetBeans.org project posted a link to our ProSim'03 Workshop article (Jensen and Scacchi 2003) where some of these ideas were initially proposed and evaluated. See http://www.netbeans.org/community/articles/UCI_papers.html.

```
sequence Set Release Date {
  ...
 iteration Update IssueZilla {
  action Report Issues To IssueZilla {
    requires { Test results }
    provides { IssueZilla entry }
    tool { Web browser }
    agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio developers }
    script {
     <br><a href="http://www.netbeans.org/issues/">Navigate to IssueZilla </a>
     <br><a href="http://www.netbeans.org/issues/query.cgi">Query IssueZilla </a>
     <br><a href="http://www.netbeans.org/issues/enter_bug.cgi">Enter issue </a> } }
 ...
```

The PML fragment (excerpt) that specifies a process step for the action, “Report Issues to IssueZilla”, corresponding to its reenactment below.

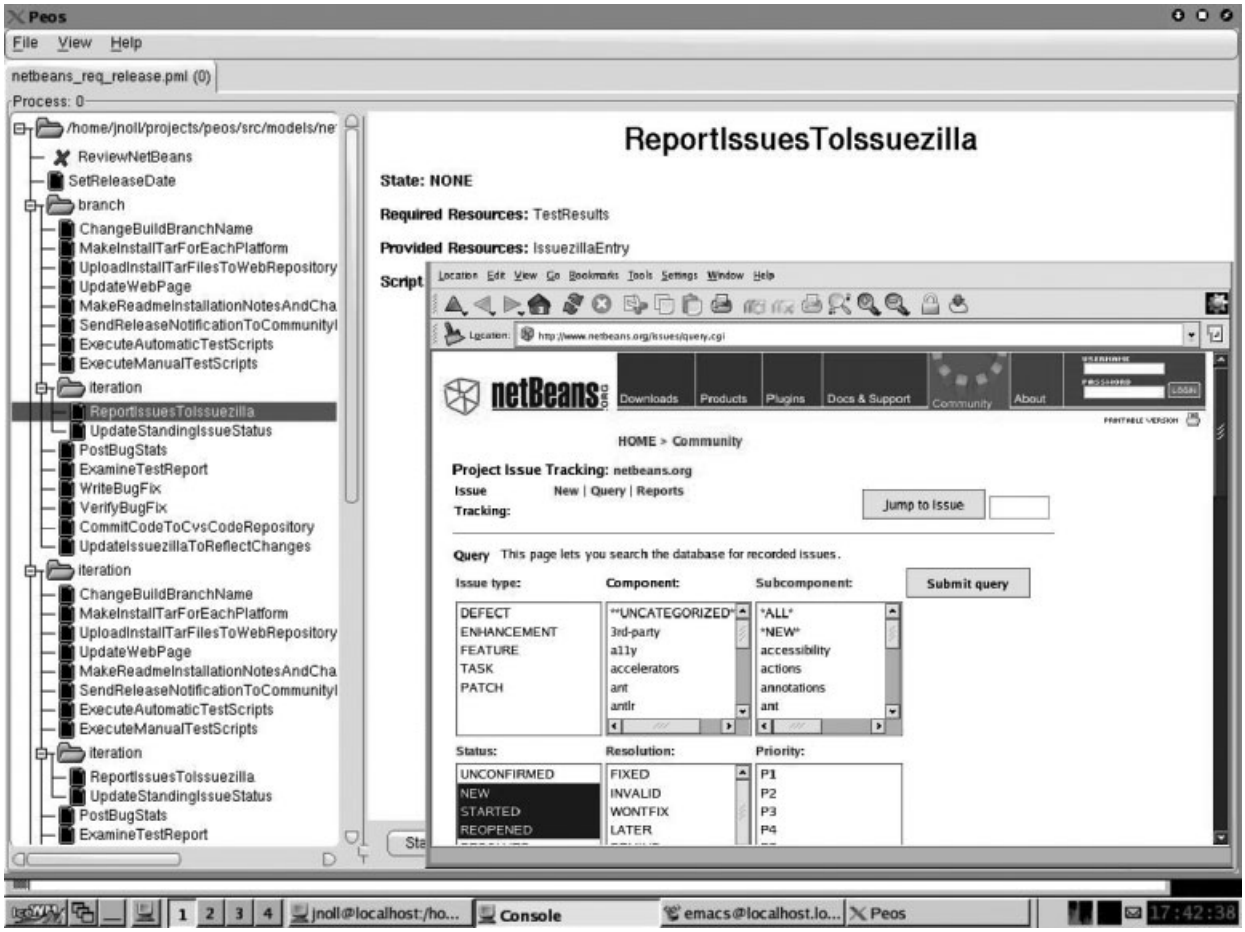


Figure 4. A step in the simulated process reenactment of the NetBeans requirements and release process (cf. Noll and Scacchi 2001)

3. MODELING PROCESSES ACROSS WEB INFORMATION INFRASTRUCTURE PROJECTS

We would like to be able to model interorganizational processes using the same techniques we use to model intraorganizational processes: top–down modeling followed by bottom–up discovery. In intraorganizational process modeling, we often know *a priori* the types of activities, artifacts, tools, and roles we are likely to see (Jensen and Scacchi 2003b). Subsequently, we can create a reference model describing how to classify their instances (Jensen and Scacchi 2003a). This guides us in our search for process data within the software development artifacts available for study. In modeling interorganizational processes used across projects in the Web information infrastructure, we have no such reference model. Instead, we first step back and examine the types of relationships that exist between organizations. Management and information systems research give us tools to characterize the types and extent of interorganizational processes. We show how to identify stakeholders (boundary agents) and objects of interaction (boundary objects), and their concerns, modeling them as a rich hypermedia. Next, we examine communication processes between organizations, as are modeled in flow diagrams, as described previously. These can be then modeled formally, and reenacted via simulation. Finally, we apply this strategy in a detailed example.

3.1. Characterizing Interoperation Among Web Information Infrastructure Projects

Successful interoperation between components of a Web information infrastructure depends on the relationships between the organizations of which it is comprised. Traditional management literature (Bluedorn *et al*. 1994) identifies six mechanisms of interorganizational interoperation or integration, which entail loose or tight coupling. These mechanisms include joint ventures, network structures, federation, cooperative agreements, trade associations, and interlocking directorates (see Table 1). Though originally devised to describe corporate and government interorganizational relationships, our choice is to apply them to provide an overall characterization of a software ecosystem for, in this case, the Web infrastructure. Unsurprisingly, the degree of coupling carries implications for the degree of process integration between these organizations. Alter (1999) defines degrees of process integration (ranging from loose to tight coupling) to include sharing a common culture, utilizing common standards, information sharing, coordination, and finally collaboration, as described in Table 2. Together, these give us a basic framework for examining the types of processes we can

Table 1. Interorganizational synchronization and stabilization mechanisms (after Bluedorn *et al*. 1994)

Interorganizational form	Example	Tightness of coupling
Joint venture	Apache, GNU foundation members	*Tightly coupled*. Two or more firms form a separate entity for a variety of strategic purposes (e.g. market power, efficiency, transfer of learning).
Network structure	System plug-in developers	*Tightly coupled*. A hub and wheel configuration with a focal firm at the hub organizing interdependencies of a complex array of firms.
Federation	Mozilla 'on-the-hook' developers	*Tightly coupled*. Established to manage and coordinate the activities of affiliated members (common in hospitals). The federation controls all or part of the management activities of the members.
Cooperative agreements	Meta-communities (e.g. JTC)	*Loosely coupled*. Arrangements between two or more firms that have strategic purposes, but do not have shared ownership.
Trade associations	Tool integration	*Loosely coupled*. Distribute trade statistics, analyze industry trends, offer legal and technical advice, and provide a platform for collective lobbying.
Interlocking directorates	NetBeans governance/community management	*Loosely coupled*. Information sharing, expertise and enhanced organizational reputation.

Table 2. Levels of business process integration (cf. Alter 1999)

Level	Example	Description
Common culture	OSSD motivations, development methods	Shared understandings and beliefs
Common standards	Data formats, communication protocols	Using consistent terminology and procedures to make business processes easier to maintain and interface
Information sharing	OSSD Web repositories	Access to each other's data by business processes that operate independently
Coordination	Meta-communities, tool integration, plug-in development	Negotiation and exchange of messages permitting separate, but interdependent, processes to respond to each other's needs and limitations
Collaboration	NetBeans, Mozilla spell-checking module development	Such strong interdependence that the unique identity of separate processes begins to disappear

expect to find in the Web infrastructure. Further, they provide the constructs of the rich hypermedia: interacting members of the Web infrastructure (stakeholders), their relationships, and the motivations of these relationships (concerns). Identification of these stakeholders, relationships, and concerns requires analysis of the interprocess communication among infrastructure projects. We address this next.

3.2. Interprocess Communication Among Web Information Infrastructure Projects

Communication between project communities provides opportunities both for integration and sources of conflict between them (Elliott and Scacchi 2003, Jensen and Scacchi 2004). We will say communication is *integrative* if it identifies compatibilities or potential compatibilities between development projects. From a process perspective, integrative communication enables external stakeholders to continue following their internal process as normal, perhaps with a small degree of accommodation. They also reinforce infrastructural processes since they do not require changes in the interoperations between communities. If the degree of accommodation or adaptation becomes too great, this can precipitate *conflictive* communication between project communities. Conflict may occur due to changes in tools or technologies shared between them, or in contentious views/beliefs for how best to structure or implement new functionality or data representations across projects. These conflicts may require extensive process articulation to adapt (cf. Scacchi and Mi 1997). Sections 4.2.1 and 4.2.2 take a closer look at process integration and conflict, followed by a discussion of how these processes are discovered and modeled.

3.2.1. Process Integration

Process integration can be direct and explicit, as in the case where NetBeans and Mozilla community members collaborated on a spell-checking module. But, it can also be indirect and implicit. For Mozilla's browser to correctly present Web artifacts, it must implement both protocols for processing Web transactions to the Apache server, and also standards for displaying content of the document or object types generated by NetBeans IDE. Similarly, the NetBeans IDE must produce artifacts and applications forms that artifact consumers, including Mozilla's browser and Apache's server, expect. The Apache server, for its part, must comply with the standard transaction protocol the Mozilla browser anticipates (e.g. HTTP/1.1) and provide Web application module support required by applications produced by the NetBeans IDE. Although these communities may not have explicitly negotiated and agreed on common data standards to be used between them, they have individually implemented standards provided and maintained by outside parties – particularly, the W3C, the World Wide Web Consortium.

These standards can be viewed as objects of interaction or *boundary objects*[4] (Star 1989, Pawlowski *et al*. 2000). Following Alter's (1999) classification, shared standards connote a low degree of process interaction between organizations in the Web infrastructure. However, other boundary objects exist, as shown in Table 3. Within OSSD project communities of the Web infrastructure, boundary objects include (a) shared beliefs and culture (Elliott and Scacchi 2003), (b) community infrastructure tools,

[4] Boundary objects are those that inhabit and span several communities of practice, as well as satisfy the informational requirements of each community (Star 1989).

Table 3. Boundary objects of the Web information infrastructure

Object type	Example
Community infrastructures	
Community culture/bylaws	Source licenses, governance style, community organizational composition
Community infrastructure tools	Defect repositories (e.g. Bugzilla, IssueZilla), collaborative development tools (e.g. WIKI, CVS, mail list managers)
Development processes	Defect discovery/submission procedures, source check-in procedures
Product infrastructure	
Product infrastructure tools	Plug-ins, modules, libraries
Development artifacts/software informalisms	Software documentation, how-to guides, design styles (e.g. P2P, client-server)
Protocols	HTTP, RPCs
Shared data formats	HTML, CGI, XML

such as defect repositories produced by other affiliated organizations (Halloran and Scherlis 2002), and (c) development processes. Additional boundary objects are found in the product infrastructure (e.g. applications program interfaces and remote procedure calls that enable data sharing and remote invocation of software modules across systems). These may take the form of software application plug-ins or modules. The Java-based Tomcat Web Server, created by the Apache community and integrated into the NetBeans IDE, is one example. They may share or coordinate development artifacts. And, as discussed, they may implement or utilize common data communication protocols and data representation formats that enable reliable communication between their tools. Although no structure is implied by the modeling paradigm, our rich hypermedia has traditionally featured development tools at the center of intraorganizational process models. In modeling the Web information infrastructure, we have expanded our view to include other types of boundary objects, as shown in Figure 5a.

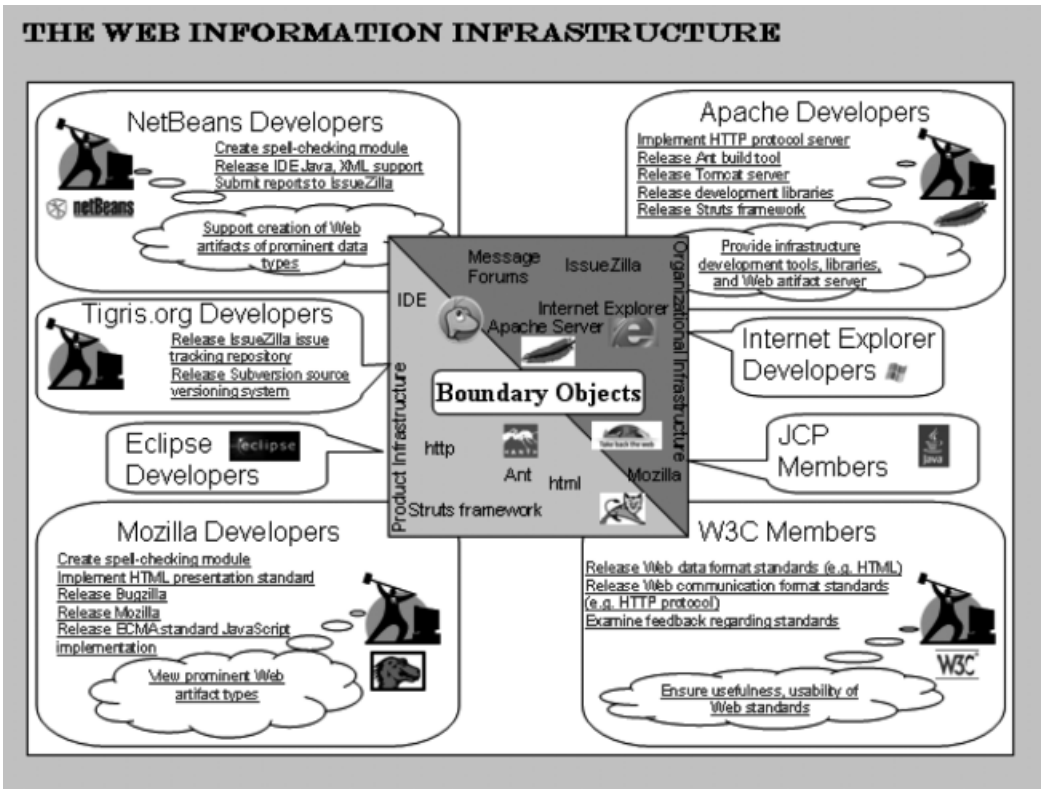
While certain boundary objects indicate a degree of interaction between processes in the Web infrastructure, it is yet unclear how this interaction plays out. As long as each member of the infrastructure adheres to these standards, they may choose to operate independently, following their individual processes as usual. However, the Web infrastructure is not a static network of interacting objects or a single coherent virtual enterprise. Commonly held standards change to meet evolving needs. Relationships between interacting software system developed by otherwise independent OSSD projects help adapt to infrastructure changes. Such relationships may require tighter coupling at the level of integration or explicit collaboration between organizational processes. By synchronizing their communication protocols and common data representations with one another through the process integration mechanisms of their choice, they stabilize the network. When an individual community varies from a standard or implements an update/revision to an existing standard, the other communities act to support it or choose to reject it. Likewise, defects in data representations or operations of one Web infrastructure software system can cause breakdowns or necessitate workarounds by the others. We look at the causes and negotiations of these conflicts next.

3.2.2. *Process Conflict*

Process conflict can precipitate or follow from process breakdown, disarticulation, or disintegration. Conflictive activities often arise from organizations competing for market share and control of the technical direction of infrastructure and shared technologies. It also arises from common and less belligerent activities, such as introducing a new version of a tool or database that other organizations depend on, requiring massive effort to incorporate. In these cases, the organization placed into conflict may simply choose to reject adopting the new tool or technology alterations, possibly selecting a suitable replacement tool/technology if the current one is no longer viable. This path was chosen by the shareware/open source image editing community infrastructure due to patent conflicts with the GIF image format in the 1990s, leading to the creation of the portable network graphics (PNG) image format standard[5].

[5] http://cloanto.com/users/mcb/19950127giflzw.html

(a)



(b)

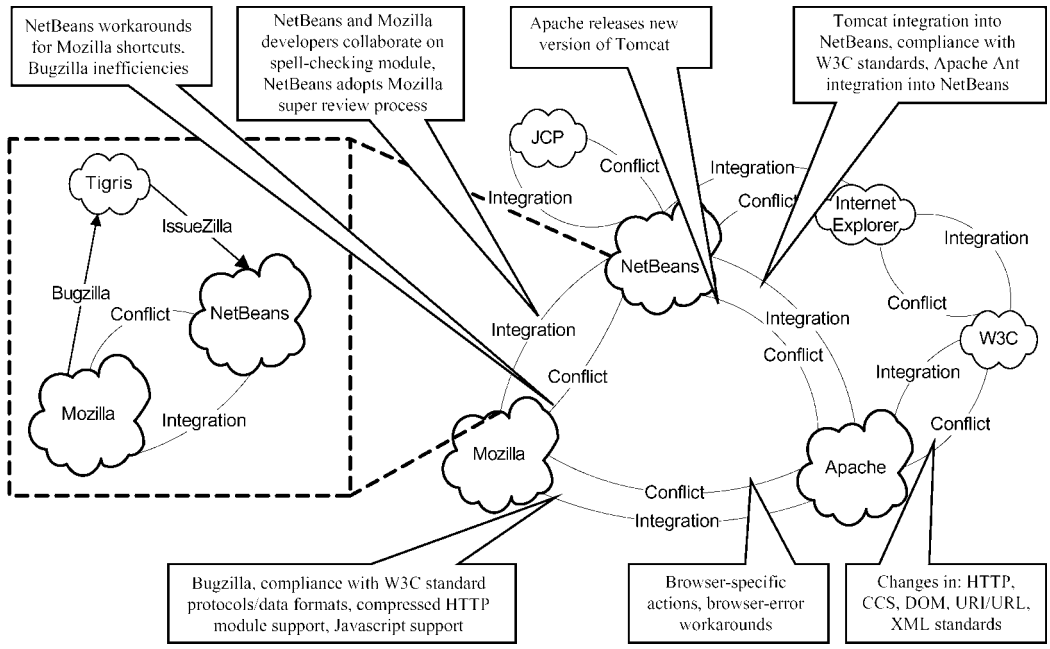


Figure 5. (a) Rich hypermedia modeling of processes spanning web information infrastructure projects. (b) Intercommunity interprocesses communication flow spanning Web information infrastructure projects. (c) Formal PML, (d) and reenactment models of processes spanning Web information infrastructure projects

(c)

```
Process Web Information Infrastructure Evolution (excerpt)
...
Sequence Mozilla Processes{   Action Release Bugzilla Defect Repository{ ...} ... }

Sequence Tigris.org Processes{   Sequence Create IssueZilla{
                        Action Download Bugzilla Sources{ ... }
                        Iteration Modify Bugzilla/IssueZilla Sources{ ... }
                        Action Release IssueZilla Issue Repository{...}
            }...
}...
Sequence NetBeans Processes{ ...
            Sequence Deploy IssueZilla issue repository{ ...          } ...
             Sequence NetBeans Development Process{
                        Action Submit Issues to IssueZilla{ ... }
                        Action Detect Inefficiency/Problems with IssueZilla{
                                    Requires { Issue reports }
                                    Provides { Problem description }
                                    Agents { NetBeans developers }
                                    Tools { HTTP Web browser implementation, IssueZilla deployment }
          Script {     <br><a href=http://qa.netbeans.org/processes/bug-handling-guidelines.html>Developers notice that IssueZilla
                      cannot track bugs across versions</a>
                      <br><a href=http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=333146>Developers
                      bring up the matter on community mailing lists/message forum</a> }
                        }
                        Action Determine Possible Workarounds/Solutions{
                                    Requires { Problem description }
                                    Provides { List of workarounds/solutions }
                                    Agents { NetBeans developers }
                                    Tools { HTTP Web browser implementation, IssueZilla deployment, developer, community discussion
message forums }
                    Script {     <br><a href=http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=389086>Discuss
                                workaround viability</a>
                                <br><a href=http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=331896>Discuss
                                workaround viability</a> }              }
```

(d)

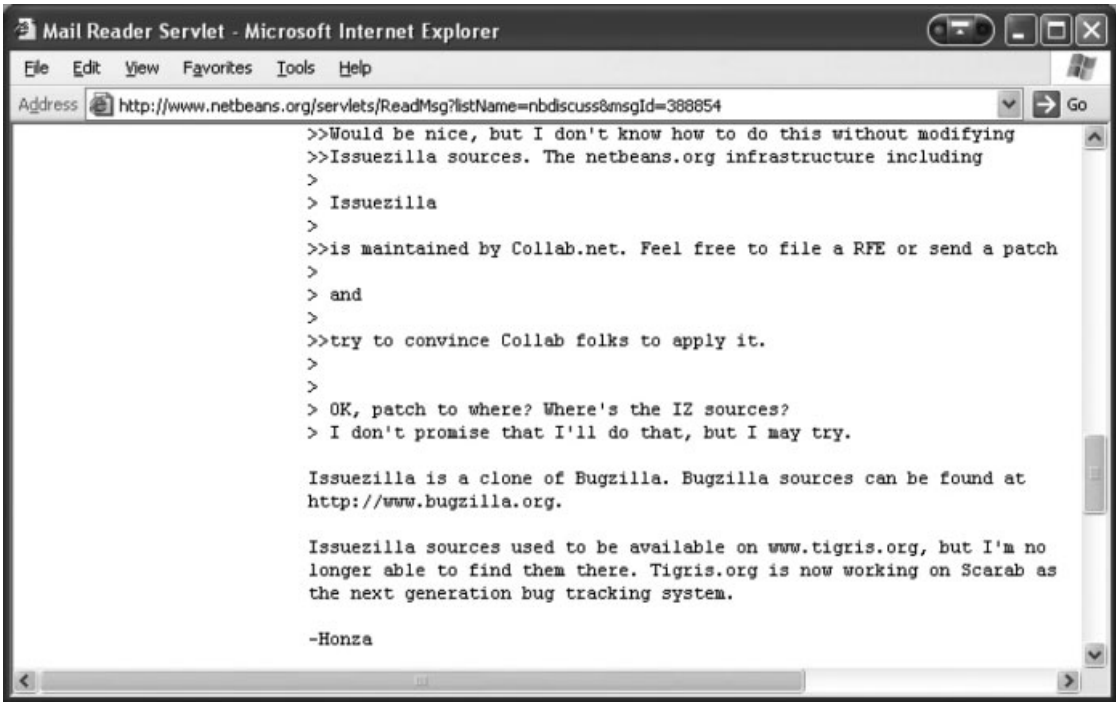


Figure 5. (*Continued*)

Conflicts across OSSD projects that get resolved are done so through collaborative means. Most typically, this occurs through the exchange of messages between participants (message threads) communicated on project discussion forums or other computer-mediated communication systems (e-mail, chat, instant messaging, etc.). Alternatively, an organization causing or resisting a tool or technology may succumb to pressure exerted by support from the rest of the infrastructure. Irreconcilable differences, if they persist and are strongly supported, can lead to unresolved conflicts (e.g. software updates that do not get implemented), incompatibilities in the interoperating software systems, or possibly to divisions in the infrastructure.

3.2.3. Discovering Interprocess Communication Across Web Information Infrastructure Projects

Though instances of direct communication between organizations within the infrastructure are visible (e.g. NetBeans and Mozilla developers collaborating on spell-checking module), indirect communication appears the more prevalent method. We see it in the form of version changelogs announcing support (and changes in support) for tools and technologies integrated into development. It may also appear in defect/feature request repositories, e-mail discourse, and community newsletters within the respective community Web sites, in addition to external news sources (e.g. slashdot.org and freshmeat.org). Communities must monitor these information sources to assess their degree of impact and whether the impact is directly or indirectly integrative or conflictive. NetBeans, for example, uses the IssueZilla bug/feature request repository developed by the Tigris community, which is, in turn, an extension of Mozilla's Bugzilla tool (see Figure 5b). Detecting indirect communications and their relationships to development process activities can be as simple as entering community specific terms, such as 'Mozilla', in the Search function input field on the NetBeans main Web page.[6] This action will search the NetBeans community Web site for instances of artifacts (e.g. project Web pages or forum postings) that contain the term Mozilla. The returned search results provide links to development artifacts or boundary objects associated with different development releases, bug reports, software module (in)compatibilities, or external (user) functionality assessments. In other cases, process communication can be nearly impossible to detect if no evidence is publicly available or is too circumstantial to validate.

3.2.4. Modeling Interprocess Communication Across Web Information Infrastructure Projects

Synchronization and stabilization of shared artifacts, data representations, and operations or transactions on them are required for a common information infrastructure to be sustained. This process is not 'owned' (Larsen and Klischewski 2004), located within, or managed by a single organization or virtual enterprise. Instead, it represents a collectively shared set of activities, artifacts, and patterns of communication that are enacted across the participating communities. Thus, it might better be characterized as an ill-defined, *ad hoc*, or one-off boundary-spanning process that differs in form with each enactment. Consequently, the form of these processes is dynamic and emergent, rather than static and recurring. Modeling such one-off processes thus must be justified, since they occur infrequently and do not reoccur. As such, our approach to modeling trades off representational detail of individual process forms, and instead uses a more abstract, low-fidelity representation (Atkinson *et al*. 2004). This is done so as to only model (or suggest) an abstract set of relationships of interaction, whose individual elements would be composed anew for each enactment.

Community communication channels (i.e. recurring patterns of communication of shared artifacts, data representations, or protocols) can be used to connect the interprocess resources flows between interoperating communities within the Web infrastructure. Each channel between communities connotes *ad hoc* processes that articulate the interoperability or interdependence of tools and technologies between them, as well as the boundary objects shared between them. The Web information infrastructure development process can therefore be characterized by the communication flow that enables integration or conflict process activities between constituent projects. Figure 5b illustrates some of the interorganizational relationships we have uncovered across the NetBeans, Mozilla, and Apache organizations, spanning many of the mechanisms of interorganizational interoperation described by Bluedorn and associates (1994), as

[6] http://www.netbeans.org/

well as the degrees of process integration outlined by Alter (1999) as a low-fidelity resource flow graph. Subsequently, these communication channels that enable interoperation and interdependence can be represented as a rich hypermedia, a low-fidelity resource flow graph, or as a low-fidelity formal process model, as shown in the examples of Figure 5. A narrative of the NetBeans IssueZilla issue tracking tool integration process depicted is these models follows.

3.3. Example: NetBeans IssueZilla Integration

The NetBeans community Web site is hosted by Sun Microsystems utilizing the commercially available collaborative development environment and Web portal software from Collab.net. Collab.net in turn utilizes the IssueZilla open source issue tracking system (sometimes also called 'issuetrack'). Thus, the NetBeans project community has a reciprocal trading relationship (see Table 1) – in this case a producer–consumer relationship – with the Tigris.org community. Tigris.org has a similar relationship with Mozilla. As such, any changes made, defects discovered, and documentation provided by Tigris.org and Mozilla may provide occasion for integration and conflict processes between NetBeans and the Tigris.org and Mozilla communities. One such occasion occurred in April of 2001. NetBeans developers noticed an inability to track issues across multiple versions and branches of the source tree[7]. This caused interorganizational coordination problems as they needed to know what technical problems existed in the previous version. In response to this and other less serious imperfections of Bug/IssueZilla system, NetBeans developers were forced to come up with workarounds until a new issue tracking system could be adopted. These workarounds involved storing the versioning information in heretofore-unused meta-data fields in each bug report, and implementing a transition plan for this new bug submission process. Unlike integration and conflict situations that arise from changes to the organizational network, the circumstances that created the need for a workaround led to a process adaptation within NetBeans that arose from an inability for existing network state to innately handle the changing needs of the NetBeans community. The adaptation process proceeded as follows.

Upon realizing the inadequacy of the issue repository, and following discussion thereof on the mailing lists[8], core developers posted an update to the bug-handling guidelines community Web page, announcing the nature of the problem and possible solutions. This was followed by discussion on the mailing lists of the desirability of the solutions presented[9]. Being an open source tool itself, some developers considered trying to modify the IssueZilla tool; however, development on it had ceased[10], the source from Bugzilla had long been forked[11], and the only available source versions were very complex[12]. In the end, the decision was made to alter the issue submission policy to use a previously unused field in the issue report to store the version-specific data as a stopgap solution, until a new defect repository was adopted. Though the inadequacy of the issue repository was reported in April 2001, the community still awaits the deployment of a new issue repository. Figure 5 provides rich hypermedia, process flow graph, and formal PML, and reenactment representations of this process.

4. DISCUSSION

The Apache, Mozilla, and NetBeans project communities are three prominent members of a larger organizational ecosystem that constitutes the Web infrastructure. In the space of software development, this ecosystem forms a development domain: the Web information infrastructure. Other prominent members of this ecosystem include OpenOffice.org, Tigris.org, Microsoft, IBM, the JCT, the W3C, and the Java Community Process (JCP). The three communities are the focus of our process modeling effort because they are developing large-scale software systems and related products through

[7] See: http://qa.netbeans.org/processes/bug-handling-guidelines.html

[8] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=333146

[9] http://www.netbeans.org/servlets/ReadMsg?listName=nbdev&msgId=79215

[10] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=388854

[11] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=389086

[12] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=331896

complex processes that integrate efforts of tens of thousands of developers with millions of users. At the same time, the ecosystem is not static. The communities, and different OSSD projects within them, rise and fade from prominence. As they increase in mass (membership) and interconnectivity, they create a sense of both gravity and inertia around them, and other organizations may seek integrative relationships. While closed source projects tend to enjoy tightly coupled integration with relatively few counterparts, OSSD communities tend towards loosely coupled interoperability with many counterparts. The effect of this is that there are likely many more organizations impinging on the ecosystem with more complex, but weaker, bindings than those of proprietary system relationship networks, which may well be sparser and rely on more stable ties enforce through contractual arrangements or partnerships.

5. CONCLUSION

In this article, we described techniques and issues in modeling software processes used within three large and interdependent open source software development communities. The software developed in these communities form an information infrastructure for creating, serving, and consuming Web artifacts. We described an approach to modeling software development processes within and across these communities, as well as issues and trade-offs that arise along the way. Our approach draws attention to the need to model such processes using informal, semistructured, and formal process modeling techniques and representations, as well as to reenact them using a process simulator. We demonstrated how development processes within these communities interact in terms of ad hoc or fragmentary processes across communities through the direct or indirect flow of development artifacts found on each community's Web sites. This helps show the potential for the use of Web-based artifacts like rich hypermedia, resource flow graphs, and semantic web/hypertext models to capture and specify the processes of OSSD projects in the Web information infrastructure. We believe this promotes a more comprehensive, multimodel understanding of the processes rendered, as well as offering insights for the application of additional process improvement tools and techniques. The results presented here suggest a need for additional work in discovering, modeling, simulating, and analyzing interorganizational software processes. Our classification framework is rooted in a more traditional, closed source software development paradigm. But, are interorganizational relationships and their associated processes changing with the involvement of nontraditional software development organizations? Do these factors differ across software ecosystems? Finally, how can these lessons be applied to devising and improving interorganizational processes? Questions such as these denote issues to be addressed in follow-on studies that seek to model software processes that span independent projects in different organizations. After processes?

ACKNOWLEDGEMENTS

The research described in this report is supported by grants #0083075, #0205679, #0205724, and #0350754 from the US National Science Foundation. No endorsement implied. Contributors to work described in this article include Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; John Georgas, Maulik Oza, Eugen Nistor, Susan Hu, Bryce Carder, Baolinh Le, Zhaoqi Chen, Veronica Gasca, Chad Ata, Michele Rousseau, and Margaret Elliott at the UCI Institute for Software Research.

REFERENCES

Alter S. 1999. *Information Systems, A Management Perspective*, 3rd edn. Addison-Wesley: Reading, MA.

Ata C, Gasca V, Georgas J, Lam K, Rousseau M. 2002. The Release Process of the Apache Software Foundation, http://www.ics.uci.edu/~michele/SP/index.html [10 January 2005]

Atkinson D, Noll J. 2003. Automated validation and verification of process models. *Proceedings of the 7th International IASTED Conference on Software Engineering and Applications*, Marina del Ray, CA.

Atkinson DC, Weeks DC, Noll J. 2004. The design of evolutionary process modeling languages. *Proc. 11th Asia-Pacific Software Engineering Conference*, Busan, Korea, 587–592.

Bluedorn A, Johnson R, Cartwright D, Barringer B. 1994. The interface and convergence of the strategic

management and organizational environment domains. *Journal of Management* **20**(2): 201–262.

Carder B, Le B, Chen Z. 2002. Mozilla SQA and Release Process, http://www.ics.uci.edu/~acarder/225/ [10 January 2005]

Choi JS, Scacchi W. 2001. Modeling and simulating software acquisition process architectures. *Journal of Systems and Software* **59**(3): 343–354.

Cusumano M, Yoffie D. 1999. Software development on internet time. *Computer* **32**(10): 60–69.

Elliott M, Scacchi W. 2003. Free software developers as an occupational community: resolving conflicts and fostering collaboration. *Proceedings of ACM International Conference on Supporting Group Work*, Sanibel Island, FL, 21–30.

Erenkrantz J. 2003. Release management within open source projects. *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, 51–55.

Fielding RT. 1999. Shared leadership in the Apache project. *Communications of the ACM* **42**(4): 44–45.

Fielding RT, Whitehead EJ, Anderson KM, Bolcher GF, Oriezy P, Taylor RN. 1998. Web based development of complex information products. *Communications of the ACM* **41**(8): 84–92.

Fowler M, Scott K. 2000. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd edn. Addison Wesley: Reading, MA.

Georgas J. 2002. *Software Process Modeling with Protégé*. University of California: Irvine, CA. 9 June, 2002. http://www.ics.uci.edu/~jgeorgas/ics225/index.htm [10 January 2005]

Halloran T, Scherlis W. 2002. High quality and open source software practices. *Proceedings of the 2nd Workshop on Open Source Software Engineering*, Orlando, FL.

Jensen C, Scacchi W. 2003a. Applying a reference framework to open source software process discovery. *Proceedings of the First Workshop on Open Source in an Industrial Context*, Anaheim, CA, 39–42.

Jensen C, Scacchi W. 2003b. Simulating an automated approach to discovery and modeling of open source software development processes. *Proceedings of ProSim'03 Workshop on Software Process Simulation and Modeling*, Portland, OR.

Jensen C, Scacchi W. 2004. Collaboration, leadership, control, and conflict negotiation in the NetBeans.org community. *Proceedings of the Fourth Workshop on Open Source Software Engineering ICSE04-OSSE04*, Edinburgh, Scotland, 48–52.

Larsen MH, Klischewski R. 2004. Process ownership challenges in IT-enabled transformation of interorganizational business processes. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii.

Lenz K, Oberweis A. 2001. Modeling interorganizational workflows with XML nets. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, Maui, Hawaii, on CD.

Mi P, Scacchi W. 1996. A meta-model for formulating knowledge-based models of software development. *Decision Support Systems* **17**(4): 313–330.

Mockus A, Fielding R, Herbsleb J. 2002. Two case studies of open source software development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology* **11**(3): 1–38.

Monk A, Howard S. 1998. The rich picture: a tool for reasoning about work context. *Interactions* **5**(2): 21–30.

Noll J, Scacchi W. 1999. Supporting software development in virtual enterprises. *Journal of Digital Information* **1**(4): http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll/[10 January 2005].

Noll J, Scacchi W. 2001. Specifying process-oriented hypertext for organizational computing. *Journal of Network and Computer Applications* **24**(1): 39–61.

Noy NF, Sintek M, Decker S, Crubezy M, Fergerson RW, Musen MA. 2001. Creating semantic web contents with protégé-2000. *IEEE Intelligent Systems* **16**(2): 60–71.

Oza M, Nistor E, Hu S, Jensen C, Scacchi W. 2002. A First Look at the NetBeans Requirements and Release Process, http://www.ics.uci.edu/cjensen/papers/FirstLook-NetBeans/ [10 January 2005]

Pawlowski S, Robey D, Raven A. 2000. Supporting shared information systems: boundary objects, communities, and brokering. *Proceedings of the Twenty First International Conference on Information Systems*, Brisbane, Queensland, Australia.

Rasch RH, Hansen JV. 1997. A design approach for analyzing interorganizational information systems. *Annals of Operations Research* **71**(1): 95–113.

Scacchi W. 2000. Understanding software process redesign using modeling, analysis and simulation. *Software Process–Improvement and Practice* **5**(2/3): 183–195.

Scacchi W. 2002. Understanding the requirements for developing open source software systems. *IEE Proceedings – Software* **149**(1): 24–39.

Scacchi W, Jensen C, Noll J, Elliott M. 2005. *Multi-Modal Modeling of Open Source Software Requirements Processes*,

Proceedings of the 1st International Conference on Open Source Systems, Genova, Italy.

Scacchi W, Mi P. 1997. Process life cycle engineering: a knowledge-based approach and environment. *International Journal of Intelligent Systems in Accounting, Finance, and Management* **6**(1): 83–107.

Shen M, Liu D. 2001. Coordinating interorganizational workflows based on process-views. *Lecture Notes in Computer Science* **2113**: 274–283.

Star SL. 1989. The structure of Ill-structured solutions: boundary objects and heterogeneous distributed problem solving. In *Distributed Artificial Intelligence*, Vol. 2., Gasser L, Huhns MN (eds.). Pitman: London, 37–54.

van der Aalst WMP. 2002a. Inheritance of interorganizational workflows to enable business-to-business E-commerce. *Electronic Commerce Research* **2**(3): 195–231.

van der Aalst WMP. 2002b. Inheritance of interorganizational workflows: how to agree to disagree without loosing control? *Information Technology and Management Journal* **2**(3): 195–231.

Experience in Discovering, Modeling, and Reenacting Open Source Software Development Processes

Chris Jensen[1] and Walt Scacchi[1]
[1]Institute for Software Research, University of California, Irvine
Irvine, CA USA 92697-3425
{cjensen, wscacchi}@ics.uci.edu

Abstract. Process discovery has been shown to be a challenging problem offering limited results. This paper describes a new approach to process discovery that examines the Internet information spaces of open source software development projects. In particular, we examine challenges, strengths, weaknesses and findings when seeking to discover, model, and re-enact processes associated with large, global OSSD projects like NetBeans.org. The longer-term goal of this approach is to determine the requirements and design of more fully integrated process discovery and modeling mechanisms that can be applied to Web-based, open source software development projects.

1 Introduction

The goal of our work is to develop new techniques for discovering, modeling, analyzing, and simulating software development processes based on information, artifacts, events, and contexts that can be observed through public information sources on the Web. Our problem domain examines processes in large, globally dispersed open source software development (OSSD) projects, such as those associated with the Apache Web server [16], Mozilla web browser [24], GNOME [9], and integrated development environments like NetBeans [18] and Eclipse [4]. The challenge we face is similar to what prospective developers or corporate sponsors who want to join a given OSSD project face in trying to understand how software development processes and activities are accomplished. As such, our efforts should yield practical results.

OSSD projects do not typically employ or provide explicit process models, prescriptions, or schemes other than what may be implicit in the use of certain OSSD tools for version control and source code compilation. In contrast, we seek to demonstrate the feasibility of automating the discovery of software process workflows via manual search and analysis methods in projects like NetBeans by analyzing the content, structure, update and usage patterns of their Web information spaces. These spaces include process enactment information such as informal task prescriptions, community and information structure and work roles, project and product development histories, electronic messages and communications patterns among project participants ([5], [26], [29]). Likewise, corresponding events that denote updates to these sources and other project repositories are also publicly

accessible. Though such ethnographic discovery approaches net a wealth of information with which to model, simulate, and analyze OSSD processes, they are limited by a lack of scalability when applied to the study of multiple OSSD development projects (cf. [13]). Subsequently, it suggests the need for a more automated approach to that can facilitate process discovery.

In our approach, we identify the kinds of OSSD artifacts (e.g. source code files, messages posted on public discussion forums, Web pages, etc.), artifact update events (e.g. version release announcements, Web page updates, message postings, etc.), and work contexts (e.g. roadmap for upcoming software releases, Web site architecture, communications systems used for email, forums, instant messaging, etc.) that can be detected, observed, or extracted across the Web. Though such an approach clearly cannot observe the entire range of software development processes underway in an OSSD project (nor do we seek to observe or collect data on private communications), it does draw attention to what can be publicly observed, modeled, or re-enacted at a distance. That is the focus of our effort.

Our approach relies on use of a process meta-model to provide a reference framework that associates these data with software processes and process models [15]. As such, we have been investigating what kinds of processing capabilities and tools can be applied to support the automated discovery and modeling of selected software processes (e.g., for daily software build and periodic release) that are common among many OSSD projects. The capabilities and tools include those for Internet-based event notification, Web-based text data mining and knowledge discovery, and previous results from process discovery studies. However, in this study, we focus on identifying the foundations for discovering, modeling, and re-enacting OSSD processes that can be found in a large, global OSSD project using a variety of techniques and tools.

2 Related Work

Event notification systems have been used in many contexts, including process discovery and analysis ([3], [30]). However, of the systems promising automated event notification, many require process performers to obtain, install, and use event monitoring applications on their own machines to detect when events occur. While yielding mildly fruitful results, this approach is undesirable for several reasons. This includes the need to install and integrate remote data collection mechanisms with local or project-specific software development tools, and it is unclear who would take on such effort within an existing OSSD project.

Prior work in process event notification has also been focused on information collected from command shell histories, applying inference techniques to construct process model fragments from event patterns (object and tool invocations) [8]. They advise that rather than seeking to discover the entire development process from enactment instances, to instead focus on creating partial process specifications that may overlap with one another. This also reflects variability in software process enactment instantiation across iterations. This imparts additional inconvenience on

project developers, and relies on her/his willingness to use the particular tools that monitor and analyze command shell events (which can become intractable when a developer uses tools or repository services from remote networked systems). By doing so, the number of process performers for whom data is collected may be reduced well below the number of participants in the project due to privacy concerns and the hassles of becoming involved. While closed source software engineering organizations may mediate this challenge by leveraging company policies, OSSD projects lack the ability to enforce adoption of such event-capture technology.

Cook and Wolf [3] utilize algorithmic and statistical inference techniques to model processes where the goal was to create a single, monolithic finite state machine (FSM) representation of the process. However, it is not entirely clear that a single FSM is appropriate for modeling complex processes. Similarly, other FSM-related process representation schemes such as Petri-Net based FUNSOFT [6] offered a wide variety of activity and state-chart diagrams. It appears however that these representations may lack scalability when applied to a process situated within a global organizational context involving multiple tools, diverse artifact types, and multiple development roles across multiple networked sites of reasonable complexity, which is typical of large OSSD projects (cf. [9]).

Last, while process research has yielded many alternative views of software process models, none has proven decisive or clearly superior. Nonetheless, contemporary research in software process technology, such as Lil Jil [2], [21] and PML [19] argues for analytical, visual, navigational and enactable representations of software processes. Subsequently, we find it fruitful to convey our findings about software processes, and the contexts in which they occur, using a mix of both informal and formal representations of these kinds [28]. We employ this practice here.

3 Problem Domain

We are interested in discovering, modeling, simulating, and re-enacting software development processes in large, Web-based OSSD projects. Such projects are often globally distributed efforts sometimes involving tens, hundreds, or thousands of developers collaborating on products constituting thousands to millions of source lines of code without meeting face-to-face, and often without performing modern methods for software engineering ([26], [27]). Past approaches have shown process discovery to be difficult, yielding limited results. However, the discovery methods we use are not random probes in the dark, nor do they simply apply prior approaches. Instead, we capitalize on contextual aids offered by the domain. Some of these include:

- Web pages, including project status reports and task assignments, may be viewed and classified (informally) as object types.
- Asynchronous communications among project participants posted in threaded email discussion lists, which address process activities indicated by process identifier keywords (e.g., design, release, testing, etc.)
- Transcripts of synchronous communication via Internet chat (cf. [5]).

- Software problem/bug and issue reports, which reveal information on software bug reporting and maintenance/repair processes
- Testing scripts and results, which highlight project-based software testing practices
- Community newsletters, which highlight project milestone events (e.g., system releases, turnover of core developers in the projects)
- Web accessible software product source code directories and repositories, which carry timestamps and other identifiers indicating when source code objects were checked in/out, and versioning information.
- Software system builds (executable binaries) and distribution packages, which are constructed and released on a periodic basis (daily, candidate (alpha, beta), and final release (distribution version)
- OSS development tools in use in an OSSD project (e.g., concurrent version system (CVS), GNU compiler collection (gcc), bug reporting (bugzilla) (cf. [10])

- OSS development resources, including other software development artifacts and process fragment descriptions (e.g., How-To guides, lists of frequently asked questions (FAQs), etc.) [26]

Each OSSD project has locally established methods of interaction, communication, leadership, and control [14], whether explicit or implicit ([26], [27]). These collaboration modes yield a high amount of empirically observable process evidence, as well as a large degree of unrelated data. However, information spaces are also dynamic. New artifacts are added, while existing ones are updated, removed, renamed and relocated, else left to become outdated. Artifact or object contents change, and project Web sites get restructured. In order to capture the history of process evolution, these changes need to be made persistent and shared with new OSSD project members. While code repositories and project email discussion archives have achieved widespread use, it is less common for other artifacts, such as instant messaging and chat transcripts, to be archived in a publicly available venue. Nonetheless, when discovering a process in progress, changes can de detected through comparison of artifacts at different time slices during the development lifecycle. At times, the detail of the changes is beneficial, and at other times, simply knowing what has changed and when is all that is important to determining the order (or control flow sequence) of process events or activity. To be successful, tools for automated process discovery must be able to efficiently access, collect, and analyze the data including areas of the project Web space such as public email/mailing list message boards, Web page updates, notifications of software builds/releases, and software bug archives in terms of changes to the OSS information space [26], [27].

To prove the viability of our process discovery approach, we demonstrate it with a case study. For this task, we examine a selected process in the NetBeans project, which is developing an open source IDE using Java technology[1]. The “requirements

[1] The NetBeans project was started in 1996, and SUN Microsystems began project sponsorship in 1998. At present, more than 60 companies are participating in the

and release" process was chosen for study because its activities have short duration, are frequently enacted, and have a propensity for available evidence that could be extracted using automated technologies. The process was discovered, modeled informally and formally, then prototyped for analysis and reenactment. The full results of our initial case study may be found elsewhere [22]. The discussion of our process discovery and modeling methods and results follows next.

4 Process Discovery and Modeling

Discovery of open source software processes relies on several data models. Firstly, we need to determine what aspects of the process we wish to discover, defined by our process meta-model. This meta-model is neither specific to our domain (OSSD processes), nor software processes. To capture OSSD software processes, we need a means of setting the problem domain within the terms of the meta-model. Such is the task of the reference model. Once this is done, we may begin looking for instances of processes within a corpus, in this case the OSSD project Web repository.

The meta-model we use is that of Mi and Scacchi [15]. It provides us with a vocabulary to describe the processes we examine. Our meta-model defines processes hierarchically: processes are composed of tasks (sets of related actions) and atomic actions. The hierarchy may be further divided (e.g. sub-processes, subtasks, and so forth) to achieve an arbitrary degree of decomposition. Each activity is defined in terms of process entities: agents that participate in the process activity, tools used by those agents in the performance of the activity, and resources that are the product of and are consumed by performance of the activity (see Figure 1). We find these to be a minimal set of entities necessary to describe a process. This meta-model is augmented with control-flow grammar in the process markup language (PML) [19] we use to formally represent software processes to show the order in which activities and instantiated.

Unlike Cook and Wolf's approach, we apply *a priori* knowledge of software development for discovering processes. Accordingly, we use a reference model [11] to help identify possible indicators (e.g., developer roles, probable tool types, input and output objects) that a given activity has occurred. We do this by creating a taxonomy of the process entities within the problem domain. Thus, we enumerate the types of tools (and resources, activities, and agents-roles) we expect to find

---

project through their developers. In 2004, the project passed the threshold of more than 100K developers contributing to the project.

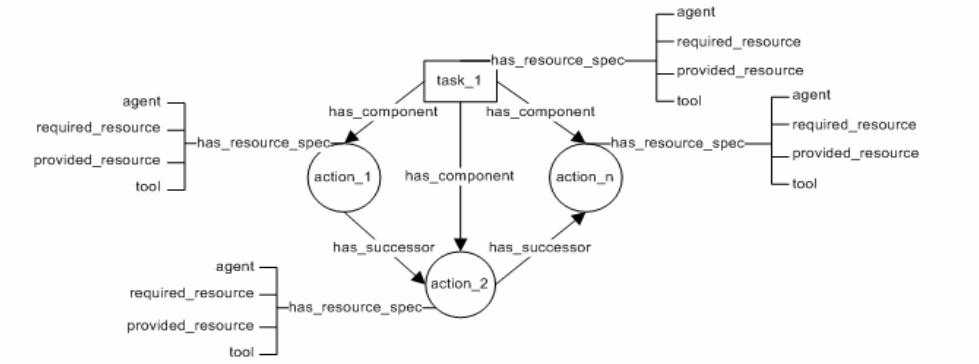


Figure 1. Software process meta-model (cf. [15]).

referenced in the project corpus (e.g. “email client”) as well as instances of those tools (e.g. “Mozilla Thunderbird”). This framework provides a mapping between the tool, resource, activity, and role names discovered in the community Web with a classification scheme of known tools, resources, activities, and roles used in open source communities. The instances are necessary for discovering process entities used within the corpus while types and genericity/hierarchy aid us in abstracting the instance data into a more general process model spanning multiple enactments.

Although we would like to achieve some degree of automation in discovering open source software development processes, it is unreasonable to assume that a complete solution is possible due to the heterogeneity of data available within and across project information corpora. Instead, we seek to automate as much as possible in order to ease the effort inherent to the task. To this end, our methodology incorporates general information gathering independent of document type, augmented by some analysis techniques specific to the type and structure of the data available. Thus we automate the tasks that are easy to automate and provide value to make the effort worthwhile. And, we do manually tasks for which automation is either too difficult or does not provide payoff to validate effort required.

We use indexing at the core to do much of the legwork of general information gathering. We use this index to identify actions, tools, resources, and agents within artifacts in the corpus. These are correlated across artifacts according to usage and update information available. While we are able to tune the reference model to contain instance values of actions, resources, and tools (e.g. “submit defect report”, “x-test-results”, and “Issuezilla,” respectively), identifying process agents by proper names a priori is not possible. Such identification requires document specific analysis techniques. These include parsers for extracting names, user handles, and email addresses from threaded mailing lists, chat logs, defect reports, and versioning repositories. Once extracted, they are looked up in the index and added to the action-tool-resource tuples already identified. Such document specific analysis may also be used to uncover heretofore-unknown instances of other process entities (i.e. actions, resources, and tools) in similar fashion and is essential to obtaining accurate

timestamps in documents that aggregate multiple software artifacts (e.g. threaded mailing lists containing multiple messages within a single system file).

Document specific analysis provides rich results with a cost. There are many types of data and standards for document structure even for a single type of data and these vary highly across OSSD projects. As a result, a broad array of tools specifically tuned for each project corpus is required to obtain rich results. Such an array is painstaking to develop, although available off-the-shelf partial solutions ease this burden. Further, integrating large result sets from multiple data sources into a single process model of any degree of formality is a complex task in itself. Our reference model can suggest process entity tuples that are related and the temporal information we are able to extract provides a timeline of activities. However learning activity control flow and asserting an activity hierarchy remain somewhat an art as opposed to a science.

The discovery of processes within a specific OSSD project begins with a cursory examination of the project Web space in order to ascertain what types of information are available and where that information might be located within the project's Web site. Structure and content of the project Web space give us an idea of what happened in terms of process actions, agents, tools, and resources, whereas artifact usage and update patterns tell us when process activities happened as noted above.

To situate the process within its organizational context, we look for modes of contribution within the development process. The modes of contribution (development roles) can be used to construct an initial set of activity scenarios, which can be described as *use cases* for project or process participation.

Though best known as a tenet of UML, use cases can serve as a notation to model scenarios of activities performed by actors in some role that use one or more tools to manipulate artifacts associated with an enterprise process or activity within it ([7], [29]). The site map also shows a page dedicated to project governance hyperlinked three layers deep within the site. This page exposes the primary member types, their roles and responsibilities, which suggest additional use cases. Unlike those found through the modes of contribution, the project roles span the breadth of the process, though at a higher level of abstraction. Each use case can encode a process fragment. In collecting use cases, we can extract out concrete actions that can then be assembled into a process description to be modeled, simulated, and enacted.

When aggregated, these use cases can be coalesced into an informal model of a process and its context rendered as a *rich hypermedia*, an interactive semi-structured

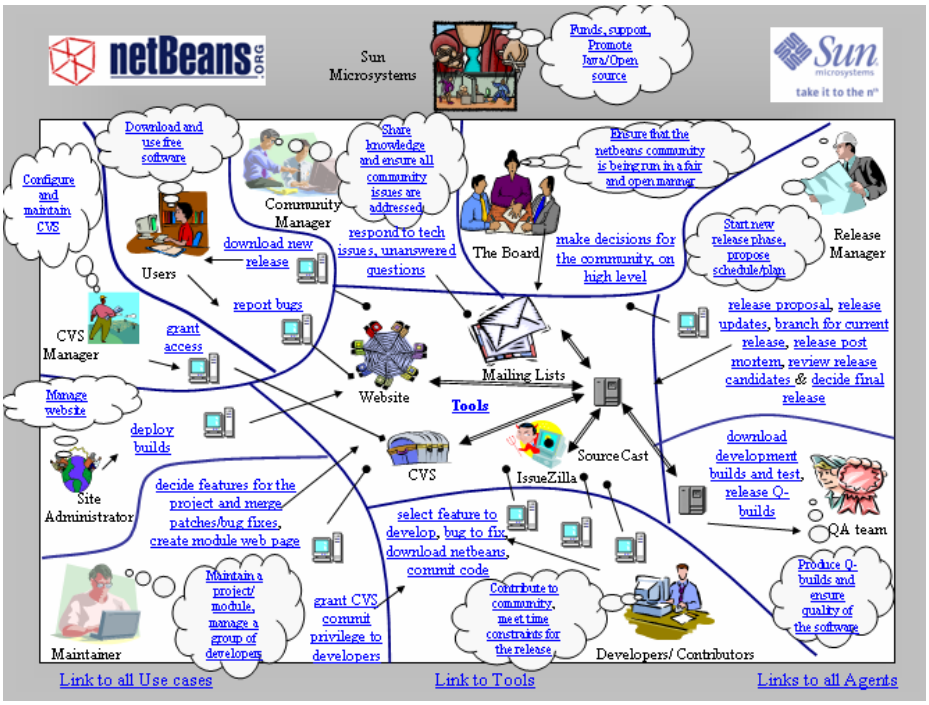


Figure 2. A hyperlinked rich hypermedia of the NetBeans requirements and release process [22]

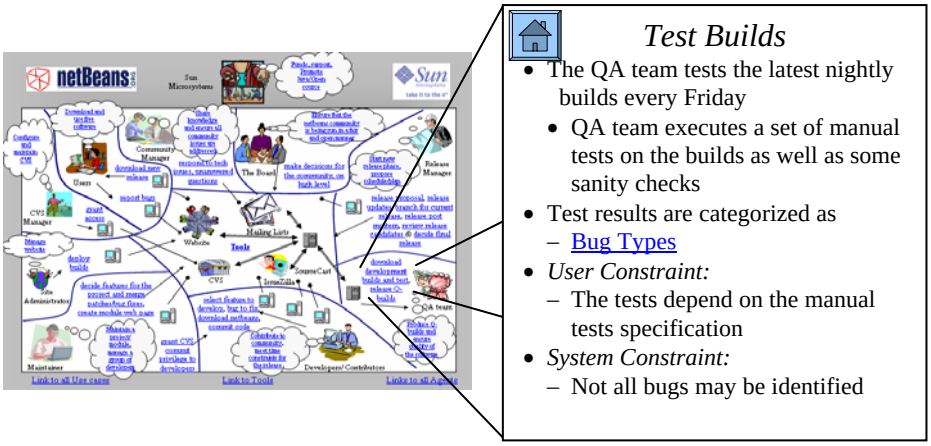


Figure 3. A hyperlink selection within a rich hypermedia presentation that reveals a corresponding use case

extension of Monk and Howard's [17] rich picture modeling construct. The rich hypermedia shown in Figure 2 identifies developer roles, tools, concerns, and artifacts of development and their interaction, which are hyperlinked (indicated as underlined phrases) to corresponding use cases and object/role descriptions (see

Figure 3). Such an informal computational model can be useful for newcomers to the community looking to become involved in development and offers an overview of the process and its context in the project, while abstracting away the detail of its activities. The use cases also help identify the requirements for enacting or re-enacting the process as a basis for validating, adapting, or improving the process.

A critical challenge in reconstructing process fragments from a process enactment instance is in knowing whether or not the evidence at hand is related, unrelated, or anomalous. Reliability of associations constructed in this fashion may be strengthened by the frequency of association and the relevance of artifacts carrying the association. If text extraction tools are used to discover elements of process fragments, they must also note the context in which are located in to determine this relevance. One way to do this is using the physical structure of the project's Web site (i.e. directory structure), as well as its logical structure (referencing/referenced artifacts). In the NetBeans quality-assurance (Q-Build) testing example, we can relate the "defects by priority" graph on the defect summary page[2] to the defect priority results from the Q-Build verification. Likewise, the defect tallies and locations correlate to the error summaries in the automated testing (XTest) results[3]. By looking at the filename and creation dates of the defect graphs, we know which sets of results are charted and how often they are generated. This in turn identifies the length of the defect chart generation process, and how often it is executed. The granularity of process discovered can be tuned by adjusting the search depth and the degree of inference to apply to the data gathered. An informal visual representation of artifacts flowing through the requirements and release process appears in Figure 4.

These process fragments can now be assembled into a formal process modeling language description of the selected processes. Using the PML grammar and process meta-model, we created an ontology for process description with the Protégé-2000 modeling tool [20]. The PML model builds from the use cases depicted in the rich hypermedia, then distills them a set of actions or sub-processes that comprise the process with its corresponding actor roles, tools, and resources and the flow sequence in which they occur. A sample PML description that results appears in Figure 5.

5 Process Reenactment for Deployment, Validation, and Improvement

Since their success relies heavily on broad, open-ended participation, OSSD projects often have informal descriptions of ways members can participate, as well as offer prescriptions for community building [26]. Although automatically recognizing and modeling process enactment guidelines or policies from such prescriptions may seem a holy grail of sorts for process discovery, there is no assurance that they accurately reflect the process as it is enacted. However, taken with the discovered process, such

[2] http://qa.netbeans.org/bugzilla/graphs/summary.html as of March 2004
[3] http://www.netbeans.org/download/xtest-results/index.html as of March 2004

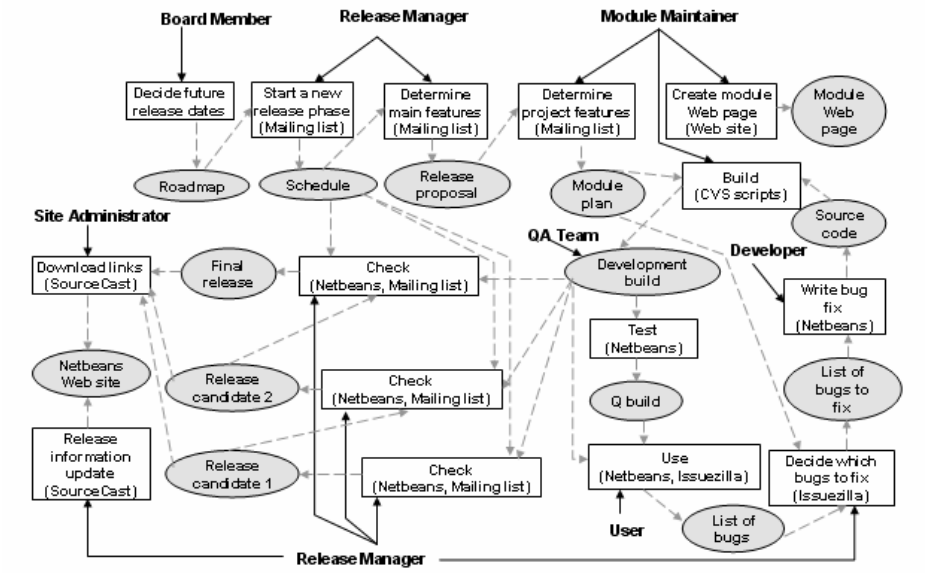


Figure 4. NetBeans Requirements and Release process flow graph [22]

```
sequence Test {
  action Execute automatic test scripts {
  requires { Test scripts, release binaries }
  provides { Test results }
  tool { Automated test suite (xtest, others) }
  agent { Sun Java Studio QA team }
  script { /* Executed off-site */ } }
action Execute manual test scripts {
  requires { Release binaries }
  provides { Test results }
  tool { NetBeans IDE }
  agent {users, developers, Sun Java Studio developers, QA team}
  script { /* Executed off-site */ } }
iteration Update Issuezilla {
  action Report issues to Issuezilla {
    requires { Test results }
    provides { Issuezilla entry }
    tool { Web browser }
    agent{users, developers, Sun Java Studio developers, QA
team}
    script {
      <br><a href="http://www.netbeans.org/issues/">Navigate to
Issuezilla </a>
      <br><a href=http://www.netbeans.org/issues/query.cgi>
Query Issuezilla </a>
      <br><ahref=http://www.netbeans.org/issues/enter_bug.cgi>
Enter issue </a> } }
```

Figure 5. A partial PML description of the testing sequence of the NetBeans release process

prescriptions begin to make it is possible to perform basic process validation and conformance analysis by reconciling developer roles, affected artifacts, and tools being used in modeled processes or process fragments (cf. [1], [23]).

As OSSD projects are open to contributions from afar, it also becomes possible to contribute explicit models of discovered processes back to the project under study so that project participants can openly review, independently validate, refine, adapt or otherwise improve their own software processes. Accordingly, we have contributed our process models and analyses of the NetBeans requirements and release process in the form of a public report hosted (and advertised) on the NetBeans.org Web site[4].

Process re-enactment allows us to recreate, simulate, or prototype process enactments by navigationally traversing a semantic hypertext (i.e., PML) representation of the process [19], [25]. These re-enactment prototypes are automatically derived from a compilation of their corresponding PML process model, and the instantiation of the complied result in a Web-based run-time (enactment) environment [19]. One step in the process modeled for NetBeans appears in Figure 6, drawn from the excerpt shown in Figure 5. In exercising repeated simulated process enactment walkthroughs, we have been able to detect process fragments that may be unduly lengthy, which may serve as good candidates for downstream process engineering activities such as streamlining and process redesign [25]. Process re-enactment also allows us, as well as participants in the global NetBeans project, to better see the effects of duplicated work. As an example, we have four agent types that test code. Users may carry out beta testing from a black box perspective, whereas developers, contributors, and SUN Microsystems QA experts may perform more in-depth white-box testing and analysis, and, in the case of developers and contributors, not merely submit a report to the IssueZilla issue tracking system,[5] but may also take responsibility for resolving it.

We are also able to detect where cycles or particular activities may be problematic for participants, and thus where process redesign may be of practical value [25]. Process re-enactment prototypes are a useful means to interactively analyze whether or how altering a process may lead to potential pitfalls that can be discovered before they lead to project failure. Over the course of constructing and executing the prototype we discovered some of the more concrete reasons that there are few volunteers for the release manager position. The role has an exceptional amount of tedious administrative tasks that are critical to the success of the project.

Between scheduling the release, coordinating module stabilization, and carrying out the build process, the release manager has a hand in almost every part of the requirements and release process. This is a good indication that downstream activities may also uncover a way to better distribute the tasks and lighten her/his load. The self-selective nature of OSSD project participation has many impacts on

---

[4] See http://www.netbeans.org/community/articles/index.html, as of May 2003.
[5] See http://www.netbeans.org/kb/articles/issuezilla.html, as of March 2004.

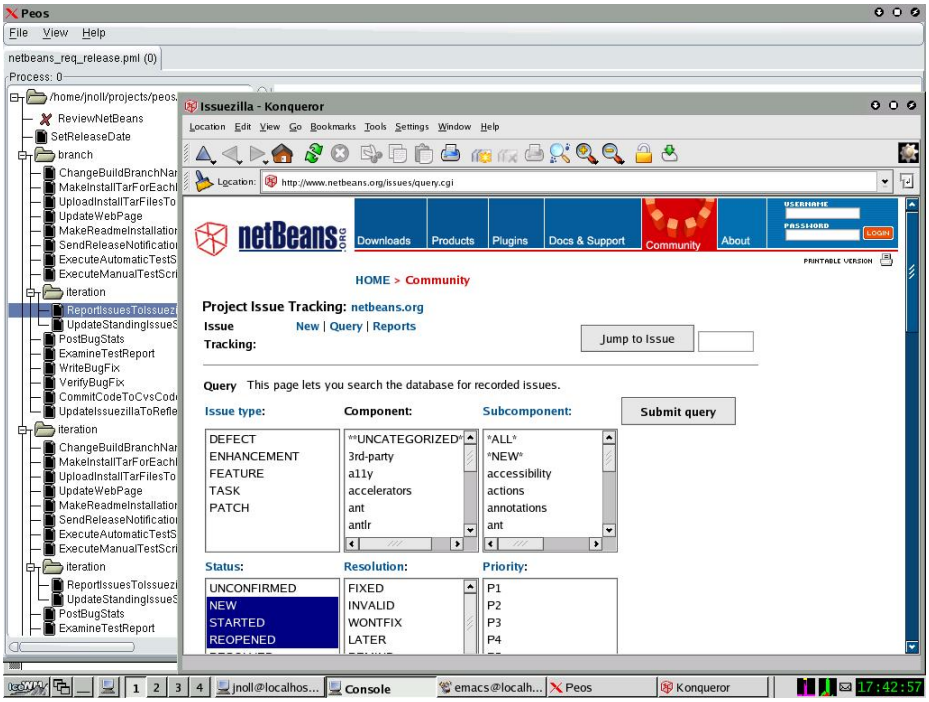


Figure 6. An action step in a re-enactment of the NetBeans requirements and release process, specified in Figure 5

their development process. If any member wishes not to follow a given process, the process enforcement is contingent on the tolerance of her/his peers in the matter, which is rarely the case in corporate development processes. If the project proves intolerant of the alternative process, developers are free to simply not participate in the project's development efforts and perform an independent software release build.

6 Conclusion

Our goal is to obtain process execution data and event streams by monitoring the Web information spaces of open source software development projects. By examining changes to the information space and artifacts within it, we can observe, derive, or otherwise discover process activities. In turn, we reconstitute process instances using PML, which provides us with a formal description of an enactable, low-fidelity model of the process in question that can be analyzed, simulated, redesigned, and refined for reuse and redistribution. But this progress still begs the question of how to more fully automate the discovery and modeling of processes found in large, global scale OSSD projects.

Our experience with process discovery in the NetBeans project, and its requirements and release process, and our case studies discovering, modeling, and reenacting processes used in the Mozilla and Apache HTTPD projects suggest that a

bottom-up strategy for process discovery, together with a top-down process meta-model, can serve as a suitable framework for process discovery, modeling and re-enactment. As demonstrated in the testing example, action sequences are constructed much like a jigsaw puzzle. We compile pieces of evidence to find ways to fit them together in order to make claims about process enactment events, artifacts, or circumstances that may not be obvious from the individual pieces. We find that these pieces may be unearthed in ways that can be executed by software tools that are guided by human assistance [12].

Our approach to discovery, modeling, and reenactment relies on both informal and formal process representations. We constructed use cases, rich pictures, flow graphs as informal but semi-structured process representations which we transformed into a formal process representation language guided by a process meta-model and support tools. These informal representations together with a process meta-model then provide a scheme for constructing formal process descriptions. Thus demonstration of a more automated process discovery, modeling, and re-enactment environment that integrates these capabilities and mechanisms is the next step in this research. Additionally, we have applied this strategy towards socio-technical OSSD process as well as processes spanning OSSD organizations and seek new process to discover, model, and reenact.

Finally, it is important to recognize that large OSSD projects are diverse in the form and practice of their software development processes. Our long-term goal in this research is to determine how to best support a more fully automated approach to process discovery, modeling and re-enactment. Our study provides a case study of a real-world process in a complex global OSSD project to demonstrate the feasibility of such an approach. Subsequently, questions remain as to which OSSD processes are most amenable to such an approach, which are likely to be of high value to the host project or other similar projects, and whether all or some OSSD projects are more/less amenable to such discovery and modeling given the richness/paucity of their project information space and diversity of artifacts. As government agencies, academic institutions and industrial firms all begin to consider or invest resources into the development of large OSS systems, then they will seek to find what the best OSSD processes are, or what OSSD practices to follow. Thus discovery and explicit modeling of OSSD processes in forms that can be shared, reviewed, modified, re-enacted, and redistributed appears to be an important topic for further investigation, and this study represents a step in this direction.

7 Acknowledgements

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

References

1. Atkinson, D.C. and Noll, J. 2003.. Automated Validation and Verification of Process Models, *Proc. 7th Intern. IASTED Conf. Software Engineering and Applications*, November.

2. Cass, A.G., Lerner, B., McCall, E., Osterweil, L. and Wise, A. 2000. Little JIL/Juliette: A process definition language and interpreter. *Proc. 22nd Intern. Conf. Software Engineering*, 754-757, Limerick, Ireland, June.

3. Cook, J. and Wolf, A.L. 1998. Discovering Models of Software Processes from Event-Based Data, *ACM Trans. Software Engineering and Methodology*, 7(3), 215-249.

4. Eclipse Web Site, 2005. http://www.eclipse.org

5. Elliott, M. and Scacchi, W., Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, Idea Publishing, Hershey, PA, 2004.

6. Emmerich, W. and Gruhn, V., FUNSOFT Nets: a Petri-Net based Software Process Modeling Language, *Proc. 6th ACM/IEEE Int. Workshop on Software Specification and Design*, Como, Italy, IEEE Computer Society Press, 175-184, 1991.

7. Fowler, M. and Scott, K. 2000. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Second Ed. Addison Wesley:

8. Garg, P.K. and Bhansali, S. 1992. Process programming by hindsight. *Proc. 14th Intern. Conf. Software Engineering*, 280-293.

9. German, D., 2003. The GNOME project: A case study of open source, global software development, *Software Process—Improvement and Practice*, 8(4), 201-215.

10. Halloran, T., and Scherlis, W. 2002. High Quality and Open Source Software Practices, *Proc. 2nd Workshop on Open Source Software Engineering*, Orlando, FL, May.

11. Jensen, C. and Scacchi, W. 2003.Applying a Reference Framework to Open Source Software Process Discovery, in *Proc. 1st Workshop on Open Source in an Industrial Context*, OOPSLA-OSIC03, Anaheim, CA October..

12. Jensen, C. and Scacchi, W. 2004. Data Mining for Software Process Discovery in Open Source Software Development Communities, submitted for publication.

13. Jensen, C. and Scacchi, W. 2005, Process Modeling across the Web Information Infrastructure, *Software Process—Improvement and Practice*, (to appear).

14. Jensen, C. and Scacchi, W. 2005b, Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Open Source Software Development Community, *Proc. 38th Hawaii Intern. Conf. Systems Sciences*, Kona, HI.

15. Mi, P. and Scacchi, W. 1996. A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330.

16. Mockus, A., Fielding, R., and Herbsleb, J., 2002.Two Case Studies in Open Source Software Development: Apache and Mozilla, *ACM Trans. Software Engineering and Methodology*, 11(3), 309-346.

17. Monk, A. and Howard, S. 1998. The Rich Picture: A Tool for Reasoning about Work Context. *Interactions*, 21-30, March-April.

18. NetBeans Web Site, 2005. http://www.netbeans.org

19. Noll, J. and Scacchi, W. 2001. Specifying Process Oriented Hypertext for Organizational Computing. *J. Network and Computer Applications* 24 39-61.

20. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W. and Musen, M.A. 2001. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2), 60-71.

21. Osterweil, L. 2003. Modeling Processes to Effectively Reason about their Properties, *Proc. ProSim'03 Workshop*, Portland, OR, May 2003.

22. Oza, M., Nistor, E., Hu, S. Jensen, C., and Scacchi, W. 2002. *A First Look at the Netbeans Requirements and Release Process*, http://www.ics.uci.edu/cjensen/papers/FirstLookNetBeans/

23. Podorozhny, R.M., Perry, D.E., and Osterweil, L. 2003, Artifact-based Functional Comparison of Software Processes, *Proc. ProSim'03 Workshop*, Portland, OR, May 2003.

24. Reis C.R. and 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

25. Scacchi, W., 2000. Understanding Software Process Redesign using Modeling, Analysis, and Simulation, *Software Process—Improvement and Practice*, 5(2/3), 183-195.

26. Scacchi, W., 2002. Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings—Software*, 149(1), 25-39.

27. Scacchi, W., 2004, Free/Open Source Software Development Practices in the Game Community, *IEEE Software*, 21(1), 59-67, Jan-Feb. 2004.

28. Scacchi, W., Jensen, C., Noll, J. and Elliott, M., 2005, Multi-Modal Modeling, Analysis, and Validation of Open Source Software Development Processes, *Proc. 1st Open Source Software Conference*, Genova, IT (to appear).

29. Viller, S., and Sommerville, I., 2000. Ethnographically Informed Analysis for Software Engineers, *Intern. J. Human-Computer Interaction*, 53, 169-196.

30. Wolf, A.L. and Rosenblum, D.S. 1993. A Study in Software Process Data Capture and Analysis. *Proc. Second Intern. Conf. on the Software Process*, 115-124.

Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes

Walt Scacchi[1], Chris Jensen[1], John Noll[1,2], and Margaret Elliott[1]
[1]*Institute for Software Research*
University of California, Irvine
Irvine, CA, USA 92697-3425
[2]*Santa Clara University*
Santa Clara, CA
Wscacchi@uci.edu

Abstract

Understanding the context, structure, activities, and content of software development processes found in practice has been and remains a challenging problem. In the world of free/open source software development, discovering and understanding what processes are used in particular projects is important in determining how they are similar to or different from those advocated by the software engineering community. Prior studies however have revealed that the requirements processes in OSSD projects are different in a number of ways, including the general lack of explicit software requirements specifications. In this paper, we describe how a variety of modeling perspectives and techniques are used to elicit, analyze, and validate software requirements processes found in OSSD projects, with examples drawn from studies of the NetBeans.org project.

Keywords: software process, process modeling, software requirements, open source software development, empirical studies of software engineering

1. Introduction

In the world of globally dispersed, free/open source software development (OSSD), discovering and understanding what processes are used in particular projects is important in determining how they are similar to or different from those advocated by the software engineering community. For example, in our studies of software requirements engineering processes in OSSD projects across domains like Internet infrastructure, astrophysics, networked computer games, and software design systems [25,26], we generally find there are no explicit software requirements specifications or documents. However, we readily find numerous examples of sustained successful and apparently high-quality OSS systems being deployed on a world-wide basis. Thus, the process of software requirements engineering in OSSD projects must be different that the standard model of requirements elicitation, specification, modeling, analysis, communication, and management [22]. But if the process is different, how is it different, or more directly, how can we best observe and discover the context, structure, activities, and content software requirements processes in OSSD projects? This is the question addressed here.

Our approach to answering this question uses multi-modal modeling of the observed processes, artifacts, and other evidence composed as an ethnographic hypermedia that provides a set of informal and formal models of the requirements processes we observe, codify, and document. Why? First, our research question spans two realms of activity in software engineering, namely,

software process modeling and software requirements engineering. So we will need to address multiple perspectives or viewpoints, yet provide a traceable basis of evidence and analysis that supports model validation. Second, given there are already thousands of self-declared OSSD projects affiliated with OSS portals like SourceForge.net and Freshmeat.net, then our answer will be constrained and limited in scope to the particular OSSD project(s) examined. Producing a more generalized model of the OSS requirements process requires multiple, comparative project case studies, so our approach should be compatible with such a goal [25]. Last, we want an approach to process modeling that is open to independent analysis, validation, communication, and evolution, yet be traceable to the source data materials that serve as evidence of the discovered process in the OSSD projects examined [cf. 15].

Accordingly, to reveal how we use our proposed multi-model approach to model requirements processes in OSSD projects, we first review related research to provide the foundational basis for our approach. Second, we describe and provide examples of the modeling modes we use to elicit and analyze the processes under study. Last, we examine what each modeling mode is good for, and what kind of analysis and reasoning it supports.

2. Related Research and Approach

There is growing recognition that software requirements engineering can effectively incorporate multi-viewpoint [7,16,22] and ethnographic techniques [22,31] for eliciting, analyzing, and validating functional and non-functional software system *product* requirements. However, it appears that many in the software engineering community treat the *process* of requirements engineering as transparent and prescriptive, though perhaps difficult to practice successfully. However, we do not know how large distributed OSSD projects perform their development processes [cf. 3].

Initial studies of requirements development across multiple types of OSSD projects [25,26] find that OSS product requirements are continuously emerging [8,9,30] and asserted after they have been implemented, rather than relatively stable and elicited before being implemented. Similarly, these findings reveal requirements practice centers about reading and writing many types of communications and development artifacts as “informalisms” [25], as well as addressing new kinds of non-functional requirements like project community development, freedom of expression and choice, and ease of information space navigation. Elsewhere, there is widespread recognition that OSSD projects differ from their traditional software engineering counterparts in that OSSD projects do not in general operate under the constraints of budget, schedule, and project management constraints. In addition, OSS developers are also end-users or administrators of the software products they develop, rather than conventionally separated as developers and/versus users. Consequently, it appears that OSSD projects create different types of software requirements using a different kind of requirements engineering process, than compared to what the software engineering community has addressed. Thus, there is a fundamental need to discover and understand the process of requirements development in different types of OSSD projects.

We need an appropriate mix of concepts, techniques, and tools to discover and understand OSSD processes. We and others have found that process ethnographies must be empirically grounded, evidence-based, and subject to comparative, multi-perspective analysis [3,7,10,15,22,25,28].

However, we also recognize that our effort to discover and understand OSSD processes should reveal the experience of software development newcomers who want to join and figure out how things get done in the project [27].

As participant observers in such a project, we find that it is common practice for newcomers to navigate and browse the project's Web site, development artifacts, and computer-mediated communication systems (e.g., discussion forums, online chat, project Wikis), as well as to download and try out the current software product release. Such traversal and engagement with multiple types of hyperlinked information provide a basis for making modest contributions (e.g., bug reports) before more substantial contributions (code patches, new modules) are offered, with the eventual possibility of proposing changing or sustaining the OSS system's architecture. These interactive experiences reflect a progressive validation of a participant's understanding of current OSSD process and product requirements [1,19]. Thus, we seek a process discovery and modeling scheme that elicits, analyzes, and validates multi-mode, hypertext descriptions of a OSSD project's requirements process. Furthermore, these process descriptions we construct should span informal through formal process models, and accommodate graphic, textual, and computationally enactable process media. Finally, our results should be in a form open to independent analysis, validation, extension, and redistribution by the project's participants.

3. Multi-Mode Process Modeling, Analysis and Validation using Ethnographic Hypermedia

An ethnographic hypermedia [4] is a hypertext that supports comparative, cross-linked analysis of multiple types of qualitative ethnographic data [cf. 28]. They are a kind of semantic hypertext used in coding, modeling, documenting, and explaining patterns of social interaction data and analysis arising in contemporary anthropological, sociological, and distributed cognition studies. The media can include discourse records, indigenous texts, interview transcripts, graphic or photographic images, audio/video recordings, and other related information artifacts. Ideally, they also preserve the form and some of the context in which the data appear, which is important for subsequent (re)analysis, documentation, explanation, and presentation.

Ethnographic studies of software development processes within Web-based OSSD projects are the focus here. Ethnographic studies that observe and explain social action through online participant observation and data collection have come to be called "virtual ethnography" [12]. Virtual ethnography techniques have been used to observe the work practices, compare the artifacts produced, and discover the processes of OSSD projects found on and across the Web [5,6,13,14,23,25,26,27]. In particular, an important source of data that is examined in such studies of OSSD projects is the interrelated web of online documents and artifacts that embody and characterize the medium and continuously emerging outcomes of OSSD work. These documents and artifacts constitute a particular narrative/textual genre ecology [29] that situate the work practices and characterize the problem solving media found within OSSD projects.

We have employed ethnographic hypermedia in our virtual ethnographic studies of OSSD projects. What does this mean, and what challenges or opportunities for requirements elicitation, analysis, and validation have emerged along the way? These questions are addressed below through examples drawn from case studies of OSSD projects, such as the NetBeans.org project [13,14], which is one of the largest OSSD projects we have studied.

As noted, the OSSD projects we study are found on the Web. Web sites for these projects consist of a network of hyperlinked documents or artifacts. Samples of sites we have studied include NetBeans.org, Mozilla.org, pache.org, and GNUenterprise.org among dozens of others. The artifacts we examine include Web pages, email discussion lists, bug reports, project to-do lists, source code files and directories, site maps, and more. These artifacts are an important part of the data we collect, examine, study, code, and analyze in order to identify OSSD work practices and development processes that arise in a given project.

We create a hypermedia of these artifacts in ways that allow us to locate the originating source(s) of data within the focal project's Web site. This allows us to maintain links to the source data materials that we observe as evidence of the process at hand, as well as to allow us to detect when these data sources have been updated or removed. (We also archive a local copy of all such data). However, we create codings, annotations, and assembled artifacts that embed hyperlinks to these documents as part of our ethnographic hypermedia. As a result, multiple kinds of ethnographic records are created including annotated artifacts, rich hypermedia pictures, and ethnographic narratives. Juxtaposed about these records are other kinds of models including a process meta-model, attributed directed graph model, process domain ontology, and a formal, computationally enactable process model. Each is described next, and each is hyperlinked into an overall ethnographic hypermedia that provides cross-cutting evidence for the observed OSS requirements processes.

Annotated artifacts

Annotated artifacts represent original software development artifacts like (publicly available) online chat transcripts that record the dialogue, discussions, and debate that emerge between OSS developers. These artifacts record basic design rationale in an online conversation form. The textual content of these artifacts can be tagged, analyzed, hyperlinked, and categorized manually or automatically [24]. However, these conversational contents also reveal much about how OSS developers interact at a distance to articulate, debate, and refine the continuously emerging requirements for the software system they are developing. For example, Elliott and Scacchi [5,6] provide conversational transcripts among developers engaged in a debate over what the most important properties of software development tools and components to use when building free software. They provide annotations that identify and bracket how ideological beliefs, social values, and community building norms constrain and ultimately determine the technical choices for what tools to use and what components to reuse when developing OSS.

Navigational rich pictures

Rich pictures [18] provide an informal graphical scheme for identifying and modeling stakeholders, their concerns, objects and patterns of interaction. We extend this scheme to form navigational rich pictures constructed as an Web-compatible hypertext image map that denotes the overall context as the composition and relationships observed among the stakeholder-roles, activities, tools, and document types (resources) found in a OSSD project. Figure 1 displays such a rich picture constructed for NetBeans.org. Associated with each relationship is a hyperlink to a *use case* [2] that we have constructed to denote an observable activity performed by an actor-role using a tool that consumes or produces a document type. An example use case is shown in Figure 2. Each other type of data also is hyperlinked to either a descriptive annotation or to a Web site/page where further information on the object type can be found.

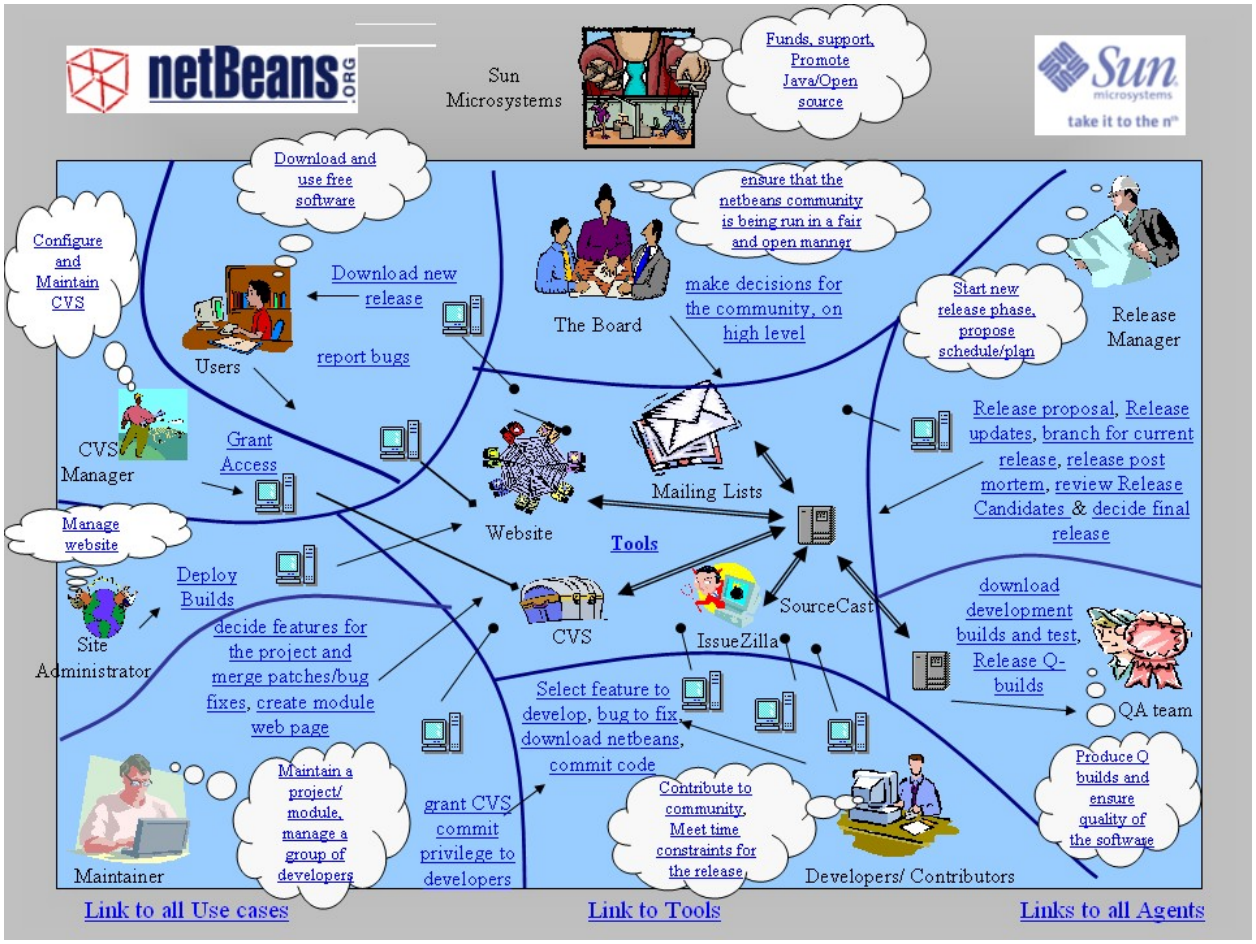


Figure 1. A rich picture image map of the requirements and release process in the NetBeans.org

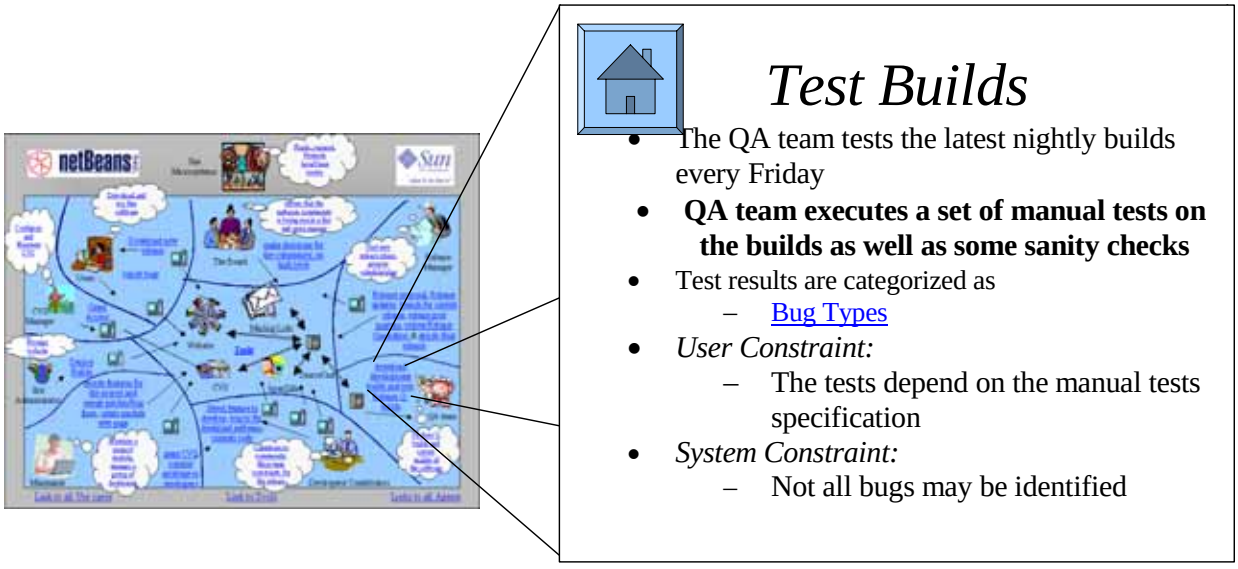


Figure 2. A hyperlink selection within a rich hypermedia presentation that reveals a corresponding use case.

Directed resource flow graph

A directed resource flow graph denotes a recurring workflow pattern that has been discovered in an OSSD project. These workflows order the dependencies among the activities that actor-roles perform on a recurring basis to the objects/resources within their project work. These resources appear as or within Web pages on an OSSD project's Web site. For example, in the NetBeans.org project, we found that software product requirements are intertwined with software build and release management. Thus, the "requirements and release process" entails identifying and programming new/updated system functions or features in the course of compiling, integrating, testing, and progressively releasing a stable composition of source code files as an executable software build version for evaluation or use by other NetBeans.org developers [5,6,23]. An example flow graph for this appears in Figure 3. The code files, executable software, updated directories, and associated email postings announcing the completion and posting the results of the testing are among the types of resources that are involved. Last, the rendering of the flow graph can serve as an image map to the online (i.e., on the NetBeans.org Web site) data sources from where they are observed.

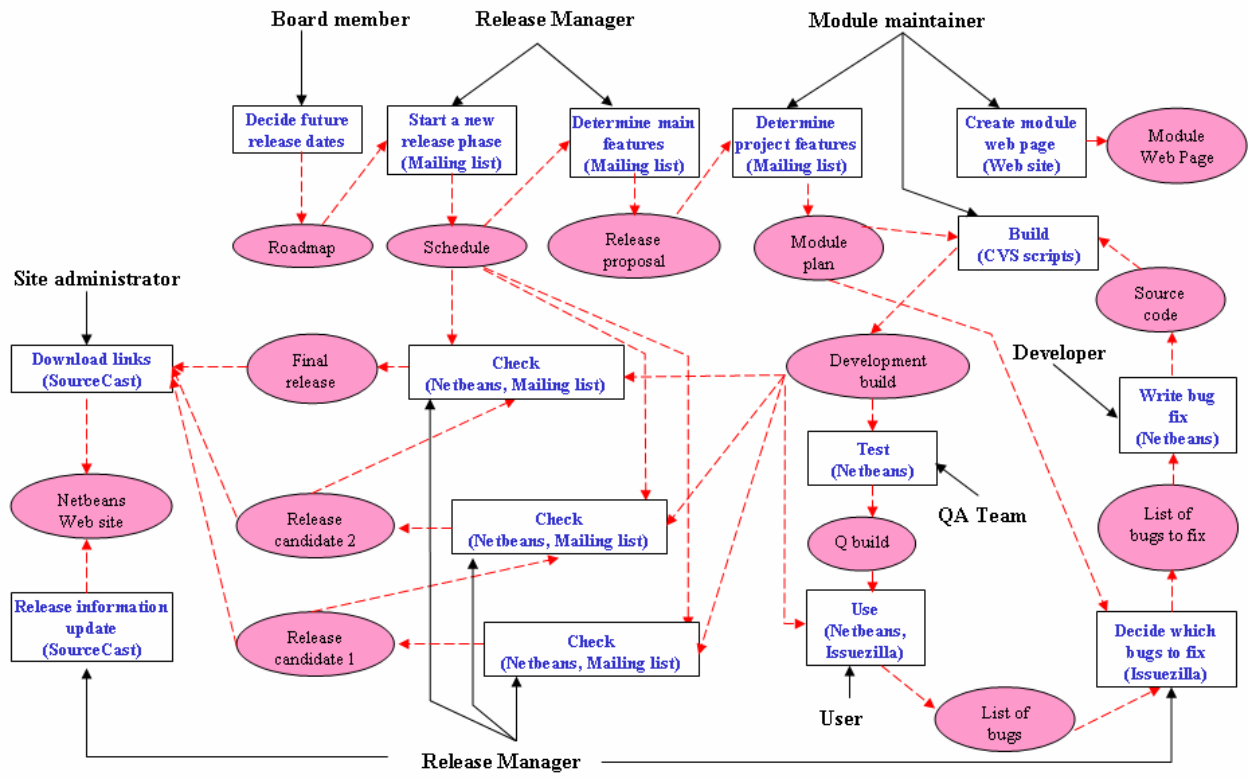


Figure 3. An attributed directed graph of the resource flow for the NetBeans.org requirement and release process. Boxes denote tasks/actions, ellipses denote resources/objects, dashed lines denote resource flows, and solid lines and labels denote agent/stakeholder roles performing tasks that transform input resources into output resources.

Process domain ontology

A process ontology represents the underlying *process meta-model* [17,20] that defines the semantics and syntax of the process modeling constructs we use to model discovered processes. It provides the base object classes for constructing the requirements process (domain) taxonomies of the object classes for all of the resource and relation types found in the rich picture and directed resource flow graph. However, each discovered process is specific to an OSSD project, and knowledge about this domain is also needed to help contextualize the possible meanings of the processes being modeled. This means that a process domain entails objects, resources or relations that may or may not be have been previously observed and modeled, so that it may be necessary to extend to process modeling constructs to accommodate new types of objects, resources, and relations, as well as the attributes and (instance) values that characterize them, and attached methods that operationalize them.

We use an ontology modeling and editing tool, Protégé-2000 [21] to maintain and update our domain ontology for OSS requirements processes. Using Protégé-2000, we can also visualize the structure of dependencies and relations [11] among the objects or resources in a semantic web manner. An example view can be seen in Figure 4. Furthermore, we can create translators that can transform syntactic form of the modeling representations into XML forms or SQL schema definitions, which enables further process modeling and tool integration options [cf. 14].

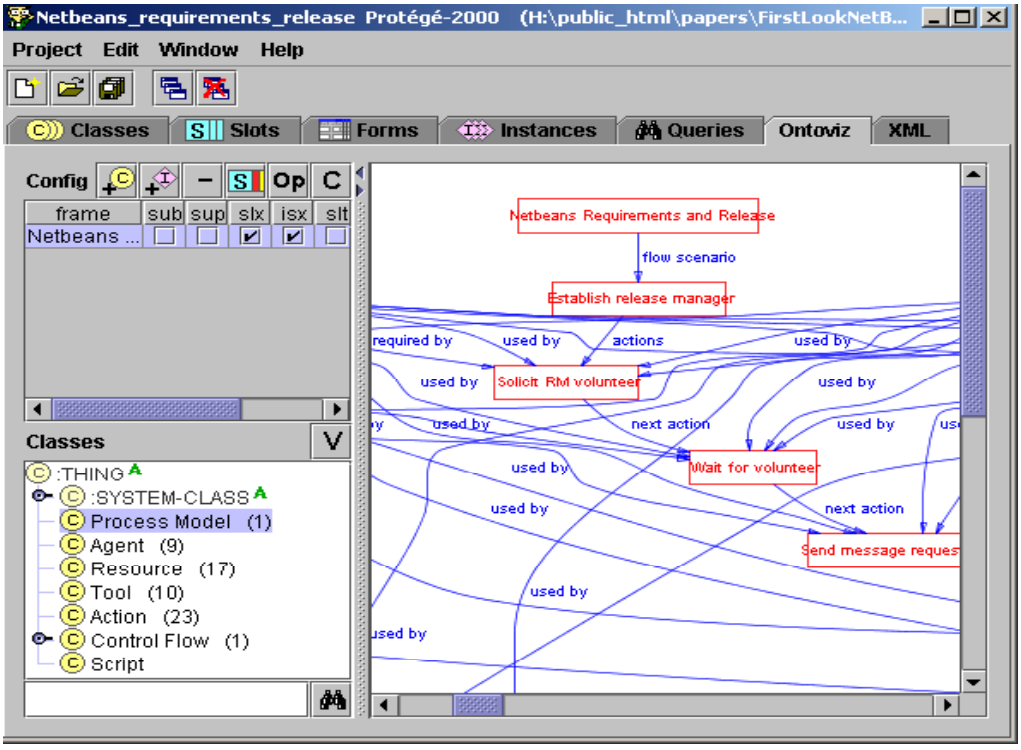


Figure 4. A view of the process domain ontology for the NetBeans.org software requirements and release process.

Formal process model and its enactment

A formal process model denotes a syntactically precise and semantically typed specification of the resource objects, flow dependencies, actor-roles, and associated tools that specifies an enactable (via interactive process-guided user navigation) hypertext representation we call an *organizational process hypertext* [20]. This semantic hypertext, and its supporting run-time environment, enables the ability to walkthrough or simulate enactment of the modeled OSSD process as a process-guided, navigational traversal across a set of process linked Web pages. The semantic hypertext is automatically rendered through compilation of the process models that are output from the ontology editor in a process modeling language called PML [20]. A PML-based model specification enables automated consistency checking at compile-time, and detection of inconsistencies at compile-time or run-time. An example of an excerpt from such a model is shown in Figure 5. The compiled version of the PML produced a non-linear sequence of process-linked Web pages, each one of which corresponds to one step in the modeled process. An example showing the result of enacting a process (action) step specified at the bottom of Figure 5 appears in Figure 6.

```
...
sequence Test {
  action Execute automatic test scripts {
  requires { Test scripts, release binaries }
  provides { Test results }
  tool { Automated test suite (xtest, others) }
  agent { Sun ONE Studio QA team }
  script { /* Executed off-site */ } }
action Execute manual test scripts {
  requires { Release binaries }
  provides { Test results }
  tool { NetBeans IDE }
  agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio developers }
  script { /* Executed off-site */ } }
iteration Update Issuezilla {
  action Report issues to Issuezilla {
    requires { Test results }
    provides { Issuezilla entry }
    tool { Web browser }
    agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio developers }
    script {
      <br><a href="http://www.netbeans.org/issues/">Navigate to Issuezilla </a>
      <br><a href="http://www.netbeans.org/issues/query.cgi">Query Issuezilla </a>
      <br><a href="http://www.netbeans.org/issues/enter_bug.cgi">Enter issue </a> } }
...
```

Figure 5. An excerpt of the formal model of the Netbeans.org requirements and release process coded in PML.

Ethnographic hypermedia narrative

An ethnographic narrative denotes the final view ethnographic hypermedia. This is an analytical research narrative that is structured as a document that is (ideally) suitable for dissemination and publication in Web-based and printed forms. It is a composite derived from selections of the preceding representations in the form of a narrative with embedded hyperlinked objects, and hyperlinks to related materials. It embodies and explains the work practices, development processes, resource types and relations, and overall project context as a narrative, hyperlinked ethnographic account that discovered at play within a given OSSD project, such as we

documented for the NetBeans requirements and release process [23]. In printed form, the narratives we have produced so far are somewhere between 1/4 to 1/15 the number of pages compared to the overall set of project-specific data (documents) at the first two levels of hyperlink connectivity; said differently, if the ethnographic report is 30 or so printed pages (i.e., suitable for journal publication), the underlying ethnographic hypermedia will correspond to a hypermedia equivalent to 120-450 printed pages.

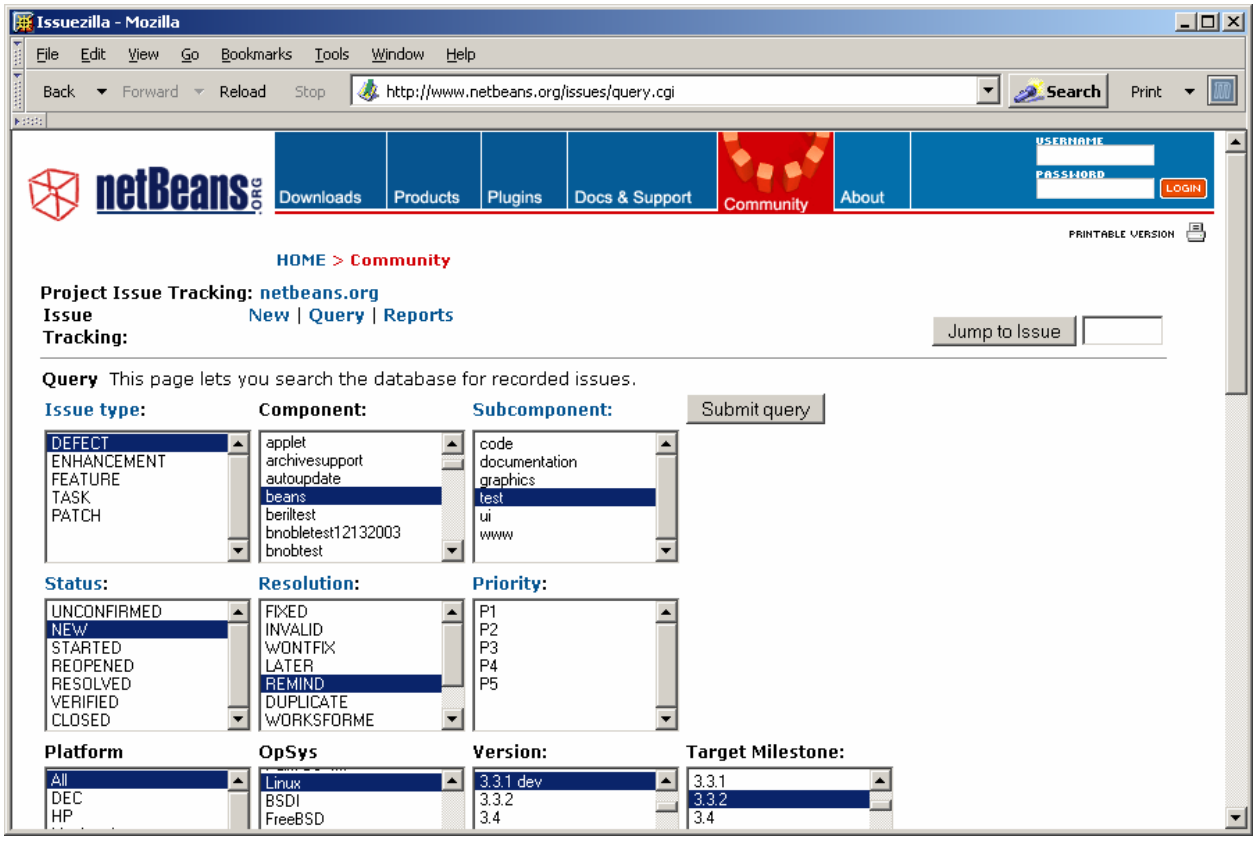


Figure 6. A screenshot displaying the result of the PML-based re-enactment of one step ("Action Report issues to Issuezilla—Query Issuezilla") in the NetBeans,org requirements and release process.

4. Discussion

We have learned a number of things based on applying our approach to requirements processes in different OSSD projects. First, no single mode of process description adequately subsumes the others, so there is no best process description scheme. Instead, different informal and formal descriptions respectively account for the shortcomings in the other, as do textual, graphic, and computationally enactable process representations. Second, incremental and progressive elicitation, analysis, and validation occur in the course of developing multi-mode requirements process models. Third, multi-mode process models are well-suited for discovery and understanding of complex software processes found in OSSD projects. However, it may not be a suitable approach for other software projects that do not organize, discuss, and perform software development activities in an online, persistent, open, free, and publicly accessible manner. Fourth, multi-mode process modeling has the potential to be applicable to the discovery and

modeling of software product requirements, although the motivation for investing such effort may not be clear or easily justified. Process discovery is a different kind of problem than product development, so different kinds of approaches are likely to be most effective.

Last, we observed that the software product requirements in OSSD projects are continually emerging and evolving. Thus, it seems likely that the requirements process in such projects is also continuously. Thus, supporting the evolution of multi-mode models of OSS requirements processes will require either automated techniques for process discovery and multi-mode update propagation techniques, or else the participation of the project community to treat these models as open source software process models, that can be continuously elicited, analyzed, and validated along with other OSSD project assets, as suggested in Figure 7, which are concepts we are currently investigating. However, it seems fair to note that ethnographic accounts are situated in time, and are not intended for evolution.

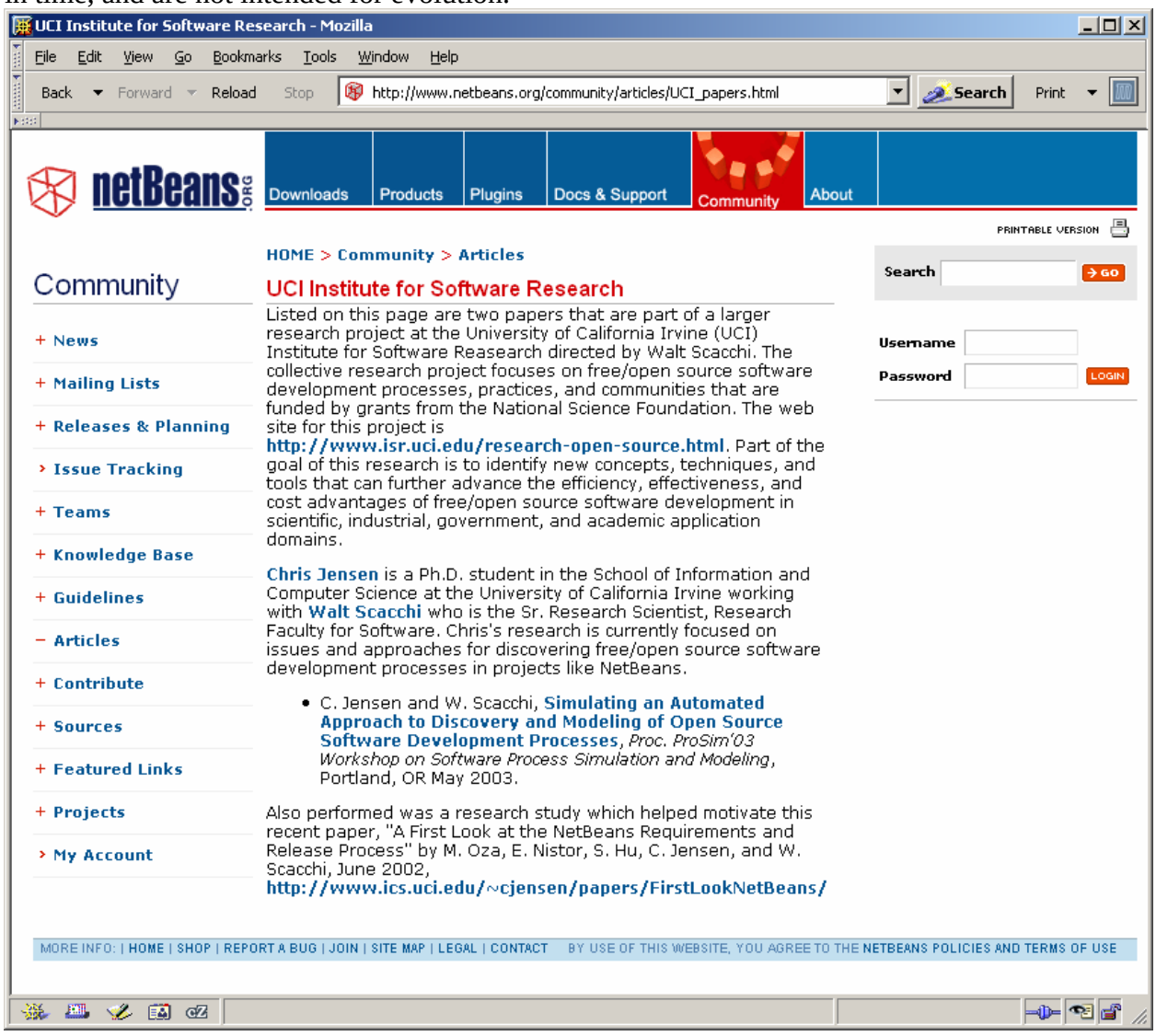


Figure 7. Getting captured and analyzed process models out for validation and possible evolution by NetBeans.org project participants.

5. Conclusion

Ethnographic hypermedia are an important type of semantic hypertext that are well-suited to support the navigation, elicitation, modeling, analysis and report writing found in ethnographic studies of OSSD processes. We have described our approach to developing and using ethnographic hypermedia in support of our studies of requirements processes in OSSD projects like NetBeans.org, where multiple modes of informal to formal representations are involved. We find that this hypermedia is well-suited for supporting qualitative research methods that associated different type of project data, with process descriptions rendered in graphic, textual and computationally enactable descriptions. We provided examples of the various kinds of hypertext-based process descriptions and linkages that we constructed in moving from abstract, informal representations of the data through a series of ever more formalized process models resulting from our studies.

6. Acknowledgements

The research described in this report is supported by grants #0083075, #0205679, #0205724, and #0350754 from the National Science Foundation. No endorsement implied. Mark Ackerman at University of Michigan, Ann Arbor; Les Gasser at University of Illinois, Urbana-Champaign; and others at ISR are collaborators on the research described in this paper.

7. References

1. Bolchini, D. and Paolini, P., Goal-Driven Requirements Analysis for Hypermedia-Intensive Web Applications, *Requirements Engineering*, 9, 85-103, 2004.

2. Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, New York, 2001.

3. Curtis, B., Krasner, H., and Iscoe, N., A Field Study of the Software Design Process for Large Systems, *Communications ACM*, 31(11), 1268-1287, 1998.

4. Dicks, B. and Mason, B., Hypermedia and Ethnography: Reflections on the Construction of a Research Approach, *Sociological Research Online*, 3(3), 1998. www.socresonline.org.uk

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

6. Elliott, M. and Scacchi, W., Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, Idea Group Publishing, Hershey, PA, 152-172, 2004.

7. Finkelstein, A.C.W., Gabbay, D., Hunter, A., Nuseibeh, B., Inconsistency Handling in Multi-perspective Specifications, *IEEE Trans. Software Engineering*, 20(8), 569-578, 1994.

8. Gans, G., Jarke, M., Kethers, S., and Lakemeyer, G., Continuous Requirements Management for Organisation Networks: A (Dis)Trust-Based Approach, *Requirements Engineering*, 8, 4-22, 2003.

9. Gasser, L., Scacchi, W., Penne, B., and Sandusky, R., Understanding Continuous Design in OSS Projects, *Proc. 16th. Int. Conf. Software & Systems Engineering and their Applications*, Paris, December 2003.

10. Glaser, B. and Strauss, A., *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine Publishing Co., Chicago, Il, 1967.

11. Grinter, R.E., Recomposition: Coordinating a Web of Software Dependencies, *Computer Supported Cooperative Work*, 12(3), 297-327, 2003.

12. Hine, C., *Virtual Ethnography*, Sage Publications, Newbury Park, CA, 2000.

13. Jensen, C. and Scacchi, W., Collaboration, Leadership, Control, and Conflict Management in the NetBeans.Org Community, *Proc. 5th Open Source Software Engineering Workshop*, Edinburgh, May 2004a.

14. Jensen, C. and Scacchi, W., Process Modeling Across the Web Information Infrastructure, *Proc. 5th Software Process Simulation and Modeling Workshop*, Edinburgh, Scotland, May 2004b.

15. Kitchenham, B.A., Dyba, T., and Jorgensen, M., Evidence-based Software Engineering, *Proc. 26th Int. Conf. Software Engineering*, 273-281, Edinburgh, Scotland, IEEE Computer Society, 2004.

16. Leite, J.C.S.P. and Freeman, P.A., Requirements Validation through Viewpoint Resolution, *IEEE Trans. Software Engineering*, 17(12), 1253-1269, 1991.

17. Mi, P. and Scacchi, W., A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330, 1996.

18. Monk, A. and Howard, S., The Rich Picture: A Tool for Reasoning about Work Context, *Interactions*, March-April 1998.

19. Narayanan, N.H. and Hegarty, M., Multimedia Design for Communication of Dynamic Information, *Int. J. Human-Computer Studies*, 57, 279-315, 2002.

20. Noll, J. and Scacchi, W., Specifying Process-Oriented Hypertext for Organizational Computing, *J. Network & Computer Applications*, 24(1), 39-61, 2001.

21. Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., and Musen, M.A., Creating Semantic Web Contents with Protégé-2000, *IEEE Intelligent Systems*, 16(2), 60-71, March/April 2001.

22. Nuseibeh, B. and Easterbrook, S., Requirements Engineering: A Roadmap, in Finkelstein, A. (ed.), *The Future of Software Engineering*, ACM and IEEE Computer Society Press, 2000.

23. Oza, M., Nistor, E., Hu, S. Jensen, C., and Scacchi, W. A First Look at the NetBeans Requirements and Release Process, http://www.ics.uci.edu/cjensen/papers/FirstLook NetBeans/, February 2004 (Original May 2002).

24. Rao, R., From Unstructured Data to Actionable Intelligence, *IT Pro*, 29-35, November 2003.

25. Scacchi, W., Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings—Software*, 149(1), 24-39, February 2002.

26. Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, Jan. 2004a.

27. Scacchi, W., Socio-Technical Interaction Networks in Free/Open Source Software Development Processes, in S.T. Acuña and N. Juristo (eds.), *Peopleware and the Software Process*, World Scientific Press, to appear, 2004b.

28. Seaman, C.B., Qualitative Methods in Empirical Studies of Software Engineering, *IEEE Trans. Software Engineering*, 25(4), 557-572, 1999.

29. Spinuzzi, C. and Zachry, M., Genre Ecologies: An Open System Approach to Understanding and Constructing Documentation, *ACM J. Computer Documentation*, 24(3), 169-181, August 2000.

30. Truex, D., Baskerville, R., and Klein, H., Growing Systems in an Emergent Organization, *Communications ACM*, 42(8), 117-123, 1999.

31. Viller, S. and Sommerville, I., Ethnographically Informed Analysis for Software Engineers, *Int. J. Human-Computer Studies*, 53, 169-196, 2000.

Breakdown, Recovery, Articulation, and Mobilization of OSSD Processes

This section contains the following four chapters that examines issues that arise when complex processes or hidden workflows associated with OSSD efforts that span one or more enterprises breakdown and must be repaired or re-articulated, in order for the process to complete, or otherwise when large-scale OSSD processes unexpectedly intersect one another.

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

Chris Jensen and Walt Scacchi, Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Software Development Community, *Proc. 38th. Hawaii Inter. Conf. Systems Science*, Waikoloa Village, HI, January 2005.

Margaret Elliott and Walt Scacchi, Mobilization of Software Developers: The Free Software Movement, revised version to appear in *Information, Technology and People.*

Walt Scacchi, Emerging Patterns of Intersection and Segmentation when Computerization Movements Interact, working paper, presented at the NSF Social Informatics Workshop, March 2005.

Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture

Margaret S. Elliott
Institute for Software Research
University of California, Irvine
Irvine, CA 92697
949 824-7202
melliott@ics.uci.edu

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697
949 824-4130
wscacchi@ics.uci.edu

August 2003
Previous version: May 2003

Revised version submitted to:
S. Koch (ed.), *Free/Open Source Software Development*, IDEA Publishing, 2004.

Acknowledgements: The research described in this report is supported by grants from the National Science Foundation #IIS-0083075, #ITR-0205679 and #ITR-0205724. No endorsement implied. Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at the Santa Clara University; Chris Jensen, Mark Bergman, and Xiaobin Li at the UCI Institute for Software Research, and also Julia Watson at The Ohio State University are collaborators on the research project that produced this chapter.

Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture

1 Introduction

Free/open source software development (F/OOSD) projects are growing at a rapid rate. The SourceForge Web site estimates 600,000+ users with 700 new ones joining every day and a total of 60,000+ projects with 60 new ones added each day. Thousands of F/OOSD projects have emerged within the past few years (DiBona, *et al*., 1999; Pavlicek, 2000) leading to the formation of globally dispersed virtual communities (Kollock and Smith, 1999). Examples of open software projects are found in the social worlds that surround computer game development; X-ray astronomy and deep space imaging; academic software design research; business software development; and Internet/Web infrastructure development (Elliott, 2003; Elliott and Scacchi, 2002; Elliott and Scacchi, 2003; Scacchi 2002a, 2002b). Working together in globally distributed virtual communities, F/OSS developers communicate and collaborate using a wide range of web-based tools including Internet Relay Chat (IRC) for instant messaging, CVS for concurrent version control (Fogel, 1999), electronic mailing lists, and more (Scacchi, 2002b).

Proponents of F/OSS claim advantages such as improved software validity, simplification of collaboration, and reduced software acquisition costs. While some researchers have examined F/OOSD using quantitative studies exploring issues like developer defect density, core team size, motivation for joining free/open source projects, and others (Koch and Schneider, 2000; Mockus *et al.*, 2000, 2002), few researchers have explored the social phenomena surrounding F/OOSD (Berquist, M. and J. Ljungberg, 2001; Mackenzie *et al.,* 2002). While the importance of understanding the culture of FOSS developers has been discussed in popular literature (Pavlicek,

2000; Raymond, 2001), no researchers have articulated the work culture of F/OOSD in a virtual organization. In this chapter, we present the results of a virtual ethnography to study the work culture and F/OOSD work processes of a *free software* project, GNUenterprise (GNUe) (http://www.gnuenterprise.org). We identify the beliefs and values associated with the free software movement (Stallman, 1999a) which are manifested into the work culture of the GNUe community and we show the importance of computer-mediated communication (CMC) such as chat/instant messaging and summary digests in facilitating teamwork, resolving conflicts, and building community.

The free software movement promotes the production of free software that is open to anyone to copy, study, modify, and redistribute (Stallman, 1999b). The Free Software Foundation (FSF) was founded by Richard M. Stallman (known as RMS in the F/OSS community) in the 1970s to promote the ideal of freedom and the production of free software, based on the concept that source code is fundamental to the furthering of computer science, and that free source code is necessary for innovation to flourish in computer science (DiBona *et al*., 1999). It is important to distinguish between the terms free software (Stallman, 1999a) and open source (DiBona *et al*., 1999). Free software differs from open source in its philosophical orientation. RMS feels that the difference is in their values, their ways of looking at the world.

> "For the Open Source movement, the issue of whether software should be open source is a practical question, not an ethical one. As one person put it, `Open source is a development methodology; free software is a social movement.' For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution. http://www.fsf.org/philosophy/free-software-for-freedom.html

A popular expression in the free software culture is "Think free speech, not free beer." The FSF promotes the use of the General Public License (GPL) for free software development as

well as other similar licenses (http://www.gnu.org/licenses/license-list.html). While the majority of open source projects use the GPL, alternative licenses suggested by the Open Source Inititative (OSI) are also available (see http://www.opensource.org).

The free software movement has spawned a number of free software projects all adhering to the belief in free software and belief in freedom of choice (http://www.gnu.org) as part of their virtual organizational culture. As with typical organizations (Martin, 1992, Schein, 1992), virtual organizations develop work cultures, which have an impact on how the work is completed. Each of these free software projects basically follow the suggested work practices outlined on the FSF Web site (see http://www.fsf.org) for initiating and maintaining a free software project. However, each project may also have cultural norms generic to their particular virtual organization. Subsequently, there is a need for better articulation of how these free software beliefs and values may influence F/OOSD. Managers and developers of F/OOSD projects would benefit from an understanding of how the culture of the free software movement influences work practices. In this chapter, we present empirical evidence from the GNUe case study of the influence that beliefs and values of the free software movement have on teamwork, tool choices, and conflict resolution in a free software development project. The results show a unique picture of one free software community and how they rely on CMC for software and documentation reviews, bug fixes, and conflict resolution. As with all qualitative research (Yin, 1992; Strauss and Corbin, 1990), we do not intend to portray a generalized view of all free software development projects. However, research has shown that many F/OOSD projects follow similar procedures (Scacchi 2002b). Future research will show how closely the GNUe work culture resembles that of other free software projects.

In the section 2 we present the GNUe project, followed by research methods in section 3. In section 4, we present background and in section 5 we discuss the GNUe virtual organizational culture with a conceptual diagram followed a description of the cases in section 6. Next we present a discussion of the data in section 7 followed by recommendations in section 8. We finish the chapter with section 9 on future research and section 10 as conclusions.

2 GNUe Project

GNUe is a meta-project of the GNU (http://www.gnu.org) Project. GNUe is organized to collect and develop free electronic business software in one location on the Web. The plans are for GNUe to consist of:

1. a set of tools that provide a development framework for enterprise information technology professionals to create or customize applications and share them across organizations;
2. a set of packages written using the set of tools to implement a full Enterprise Resource Planning system; and
3. a general community of support and resources for developers writing applications using GNUe tools. The GNUe Web site advertises it as a "Free Software project with a corps of volunteer developers around the world working on GNUe project."

GNUe is an international virtual organization for software development (Crowston and 2002; Noll and Scacchi, 1999) based in the U.S. and Europe. This organization is centered about the GNUe Web portal and global Internet infrastructure that enables remote access and collaboration. As of the writing of this paper, GNUe contributors consist of 6 core maintainers (co-maintainers who head the project); 18 active contributors; and 18 inactive contributors. The 6 core maintainers share various tasks including the monitoring of the daily IRC, accepting bug

fixes to go into a release, testing software, documentation of software, etc. Another task for these core maintainers appears to be that of trying to resolve conflicts and answering questions regarding GNUe. For the duration of the IRC logs that we studied, several core maintainers were on the IRC almost the entire day. Companies from Austria, Argentina, Lithuania, and New Zealand support paid contributors, but most of the contributors are working as non-paid participants.

3 Research Methods

This ongoing ethnography of a virtual organization (Hine, 2000; Olsson, 2000) is being conducted using the grounded theory approach (Strauss and Corbin, 1990) with participant-observer techniques. The sources of data include books and articles on OSSD, instant messaging (Herbsleb and Grinter, 1999, Nardi *et al.*, 2000) transcripts captured through IRC logs, threaded email discussion messages, and other Web-based artifacts associated with GNUe such as Kernel Cousins(summary digests of the IRC and mailing lists – see http://kt.zork.net). This research also includes data from email and face-to-face interviews with GNUe contributors, and observations at Open Source conferences. The first author spent over 100 hours studying and perusing IRC archives and mailing list samples during open and axial coding phases of the grounded theory. During open coding the first case study presented here was selected as representative of the strong influence of cultural beliefs on GNUe software development practices. The selection of cases was aided by the indexing of each Kernel Cousin into sections labeled with a topic. For example, we read through all Kernel Cousins looking mainly at the indices only and found the following title "Using Non-Free Tools for Documentation" in (http://kt.zork.net/GNUe/gnue20011124_4.html). Hyperlinks from this cousin pointed us to a similar case where non-free tools were being used for documentation of code. The third case was found by coding the last file in the three day series for the case two debate. In the third case,

a newcomer asks for help regarding the use of GNUe and we show how cooperation and community building are facilitated by the use of IRC.

The initial research questions that formed the core of the grounded theory are:

1) How do people working in virtual organizations organize themselves such that work is completed?
2) What social processes facilitate open source software development?
3) What techniques are used in open source software development that differ from typical software development?

We began this research with the characterization of open source software communities as communities of practice. A community of practice (COP) is a group of people who share similar goals, interests, beliefs, and value systems in a common domain of recurring activity or work (Wenger, 1998). An alternative way of viewing groups with shared goals in organizations is to characterize them as organizational subcultures (Trice and Beyer, 1993; Schein, 1992; Martin, 2002). As the grounded theory evolved, we discovered rich cultural beliefs and norms influencing "geek" behavior (Pavlicek, 2000). This led to us to the characterization of the COPs as virtual organizations having organizational cultures.

We view culture as both objectively and subjectively constrained (Martin, 2002). In a typical organization, this means studying physical manifestations of the culture such as dress norms, reported salaries, annual reports, and workplace furnishings and atmosphere. In addition, subjective meanings associated with these physical symbols are interpreted. In a virtual organization, these physical cultural symbols are missing, so we focus on unique types of

accessible manifestations of the GNUe culture, such as Web site documentation and downloadable source code. We use the grounded theory approach to build a conceptual framework and develop a theory regarding the influence of organizational culture on software development in a free software project (Strauss and Corbin, 1990). Data collection includes the content analysis of Web site documents; IRC archives; mailing lists; kernel cousins; email interviews; and observations and personal interviews from open source conferences.

During the open coding, we interpreted books and documents as well as Web site descriptions of the OSSD process. We discovered strong cultural overtones in the readings and began searching for a site to apply an analysis of how motivations and cultural beliefs influenced the social process of OSSD. We selected GNUe as a research site because it exemplified the essence of free software development providing a rich picture of a virtual work community with a rapidly growing piece of downloadable free software. The GNUe Web site offered access to downloadable IRC archives and mailing lists as well as lengthy documentation - all facilitating a virtual ethnography. We took each IRC and kernel cousin related to the three cases and applied codes derived from the data (Strauss and Corbin, 1990). We used a text editor to add the codes to the IRC text logs using [Begin and End] blocks around concepts we identified such as "belief in free software". In this way, we discovered the relationships shown in Figure 1. During the axial coding phase of several IRC chat logs, mailing lists and other documentation, we discovered relationships between beliefs and values of the work culture and manifestations of the culture. In the next section we discuss the organizational culture perspective and studies relating to conflict resolution in cyberspace.

4 Background

In this section, we discuss the organizational culture perspective that is used to characterize the work culture of the virtual organization, GNUe. Next we discuss literature related to conflict resolution in virtual communities.

4.1 Organizational Culture Perspective

Popular literature has described open source developers as members of a "geek" culture (Pavlicek, 2000) notorious for nerdy, technically savvy, yet socially inept people, and as participants in a "gift" culture (Berquist and Ljungberg, 100; Raymond, 2001) where social status is measured by what you give away. However, no empirical research has been conducted to study FOSS developers as virtual organizational cultures (Martin, 2002; Schein, 1992) with beliefs and values that influence decisions and technical tool choices. Researchers have theorized the application of a cultural perspective to understand IT implementation and use (Avison and Myers, 1995), but few have applied this to the workplace itself (Dube´ and Robey, 1999; Elliott, 2000).

Much like societal cultures have beliefs and values manifested in norms that form behavioral expectations, organizations have cultures that form and give members guidelines for "the way to do things around here." An organizational culture perspective (Martin, 2002; Schein, 1992; Trice and Beyer, 1993) provides a method of studying an organization's social processes often missed in a quantitative study of organizational variables. Organizational culture is a set of socially established structures of meaning that are accepted by its members (Ott, 1989).

The substances of such cultures are formed from ideologies, the implicit sets of taken-for-granted beliefs, values, and norms. Members express the substance of their cultures through the use of cultural forms in organizations -- acceptable ways of expressing and affirming their beliefs,

values and norms. When beliefs, values, and norms coalesce over time into stable forms that comprise an ideology, they provide causal models for explaining and justifying existing social systems. In a virtual organization, cultural beliefs and values are manifest in norms regarding communication and work issues (if a work-related community like OSSD) and in the form of electronic artifacts – IRC archives, mailing list archives, and summary digests of these archives as Kernel Cousins. Most organizational culture researchers view work culture as a consensus-making system (Ott, 1989; Trice and Beyer, 1993; Schein, 1992). In the GNUe study, we apply an integration perspective (Martin, 2002) to the GNUe community to show how beliefs and values of the free software movement tie the virtual organization together in the interests of completing the GNUe free software project (See Elliott and Scacchi, 2003 for a detailed report of the GNUe study). We present the GNUe virtual organization as a subculture of the FSF inculcating the beliefs and values of the free software movement into their everyday work.

4.2 Conflict Resolution in Virtual Communities

Researchers have attempted to understand conflict resolution in virtual communities (Kollock and Smith, 1996; Smith, 1999) in the areas of online communities and in the game world. Many others have studied conflict resolution in common work situations such as computer-supported cooperative work (CSCW) (Easterbrook, 1993). For our purposes, we are interested in virtual communities and how they resolve conflicts so this discussion does not include studies on conflict management tools.

Smith (1999) studied conflict management in MicroMUSE, a game world dedicated to the simulation and learning about a space station orbiting the earth. There were two basic classes of participants: users and administrators. Disputes arose in each group and between the two groups regarding issues like harassment, sexual harassment, assault, spying, theft, and spamming. These

problems emerged due to the different meanings attributed to MicroMUSE by its players and administrators and due to the diverse values, goals, interests, and norms of the group. Smith concluded that virtual organizations have the same kinds of problems and opportunities brought by diversity as real organizations do, and that conflict is more likely, and more difficult to manage than in real communities. Factors contributing to this difficulty are: wide cultural diversity; disparate interests, needs and expectations; nature of electronic participation (anonymity, multiple avenues of entry, poor reliability of connections and so forth); text-based communications; and power asymmetry among users. On the contrary, in our GNUe study, we found that text-based communications via the archival text (IRC and Kernel Cousins) enabled the conflict resolution.

Kollock and Smith (1996) explored the implications of cooperation and conflict in Usenet groups emphasizing the importance of recognizing the free-rider problem. In a group situation where one person can benefit from the product or resource offered by others, each person is motivated not to contribute to the joint effort, instead free-riding on others' work. The authors do a detailed analysis of this free-rider problem and give suggestions for how to avoid it in Usenet groups. For example, they suggest bandwidth be used judiciously, posting useful information and refraining from posting inappropriate information as a way to better manage bandwidth. Success on a Usenet group also depends on its members following cultural rules of decorum. We explore the topic of following cultural rules in the next section by presenting the conceptual framework of the GNUe study.

5 Conceptual Diagram of GNUe Virtual Organizational Culture

The substance of a culture is its ideology – shared, interrelated sets of emotionally charged beliefs, values and norms that bind people together and help them to make sense of their worlds (Trice and Beyer, 1993). While closely related to behavior, beliefs, values, and norms are unique concepts as defined below (Trice and Beyer, 1993):

- **Beliefs** – Express cause and effect relations (i.e. behaviors lead to outcomes).
- **Values** – Express preferences for certain behaviors or for certain outcomes.
- **Norms** – Express which behaviors are expected by others and are culturally acceptable

As members of the FSF, free software developers share an ideology based on the belief in free software and the belief in freedom of choice. These beliefs are espoused in the literature on free software (Williams, 2002). The values of cooperative work and community are inferred from this research. Figure 1 shows a conceptual diagram of the GNUe case study. The causal conditions consist of the beliefs (free software and freedom of choice) and the values (cooperative work and community). The phenomenon is the free software development process – its formal and informal work practices. The interaction/action occurs on the IRC and mailing lists. It consists of 1) the conflict over the use of a non-free tool to create a graphic diagram of the emerging GNUe system design, 2) the conflict over the use of a non-free tool to create GNUe documentation. The consequences are: 1) building community; 2) resolution of conflicts with a reinforcement of the beliefs; and 3) teamwork is strengthened. The beliefs, values, and norms are described below; the consequences are presented in the Discussion section.

5.1.1 Belief in Free Software

The belief in free software appears to be a core motivator of free software developers. GNUe developers extol the virtues of free software on its Web site and in daily activity on the IRC logs. The FSF Web site has many references to the ideological importance of developing and maintaining free software (See http://www.fsf.org). This belief is manifested in electronic artifacts such as the Web pages, source code, GPL license, software design diagrams, and accompanying articles on their Web site and elsewhere. The data analysis of the GNUe cases

showed that this belief varies from moderate to strong in strength. For example, those who have a strong belief in free software refuse to use any form of non-free software (such as a commercial text editor) for development purposes. The variation in strength of this variable becomes the focal point of case two.

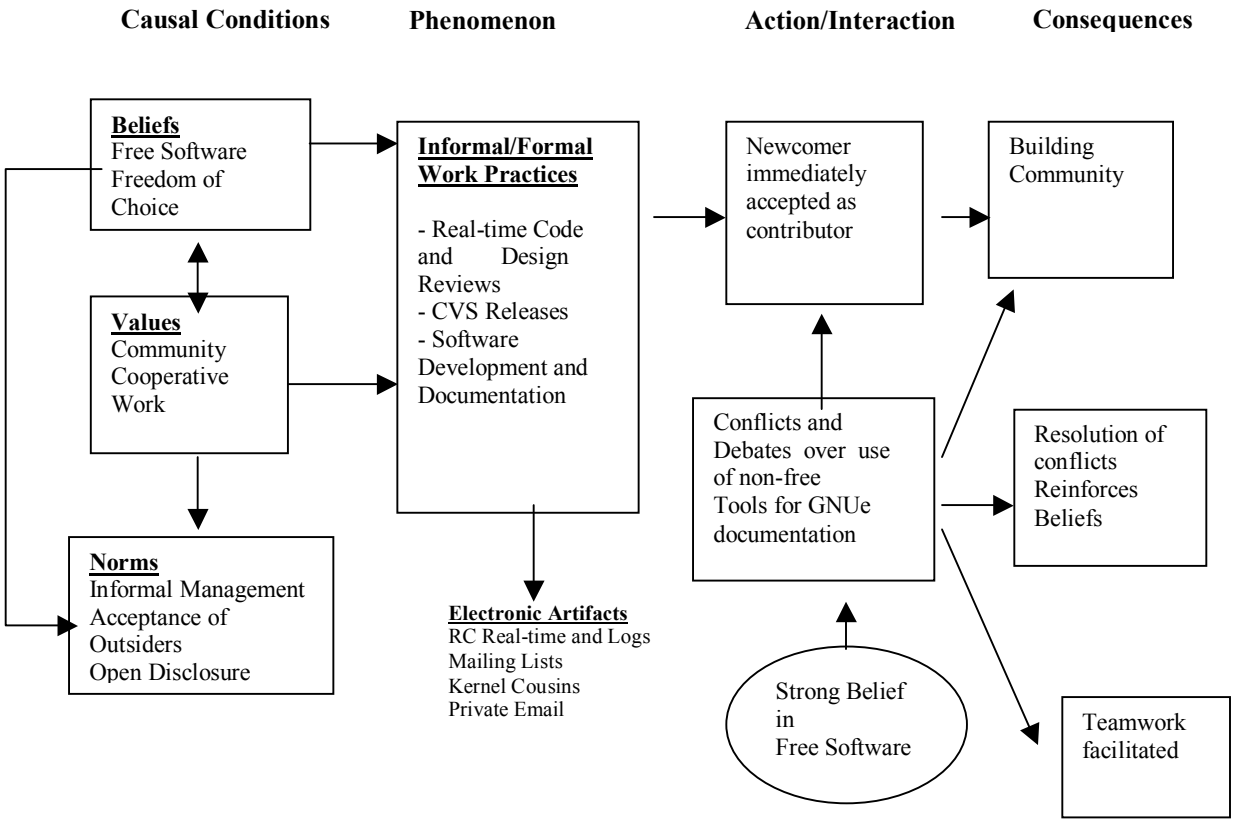


Figure 1. **Conceptual Diagram of Variables**

5.1.2 Belief in Freedom of Choice

Open source software developers are attracted to the occupation of OSSD for its freedom of choice in work assignments. Both paid and unpaid GNUe participants to some degree can select the work they prefer. This belief is manifested in the informal methods used to assign or select work in an open source project. During an interview with one of the core contributors of GNUe, Derek, at a LinuxWorld conference in August 2002, we asked how assignments were made and monitored. Derek answered with:

"The number one rule in free software is 'never do timelines or roadmaps'."

The belief in freedom of choice also refers to the ability to select the tool of choice to develop free software. Some OSS developers believe that a mix of free versus non-free software tools is acceptable when developing free software, while others adhere to the belief in free software only.

5.1.3 Value in Community

The beliefs in free software and freedom of choice foster a value in community building as part of routine work. This value is evident in the IRC archives when newcomers join GNUe offering suggestions, or pointing out bugs, and GNUe contributors quickly accept them as part of the community. For example, when frequent contributors (insiders) have a problem with procedures or code related to free versus non-free software, the maintainers rally around the insider trying to convince him that a temporary use of non-free software is OK.

5.1.4 Value in Cooperative Work

The GNUe community's beliefs in free software and freedom of choice combined with the value in community foster a value in cooperative work. As with previous researchers (Easterbrook, 1993; Kollock and Smith, 1996; Smith, 1999), our results indicate that conflict arises during the course of cooperative work. GNUe contributors work cooperatively to resolve conflicts through the use of IRC and mailing lists.

5.1.5 Open Disclosure

Open disclosure refers to the open content of the GNUe Web site including the software source code, documentation, and archived records of IRC, kernel cousins, and mailing list interchanges. The GNUe contributors join others online via IRC on a daily basis and record the conversations for future reference. All documentation and source code are easily downloaded from the GNUe Web site and user criticism is welcomed by frequent GNUe maintainers.

5.1.6 Informal Management

The entire GNUe virtual organization is informal. There is no lead organization or prime contractor that has brought together the alliance of individuals and sponsoring firms as a network virtual organization. It is more of an emergent organizational form where participants have in a sense discovered each other, and have brought together their individual competencies and contributions in a way whereby they can be integrated or made to interoperate (Crowston and Scozzi, 2002). The participants come from different small companies or act as individuals that collectively move the GNUe software and the GNUe community forward. Thus, the participants self-organize in a manner more like a meritocracy (Fielding, 1999). There is a flow to the work determined by participants' availability.

5.1.7 Immediate Acceptance of Outsider Critiques

In the GNUe organization, outsiders who have not visited the GNUe IRC before, can easily join the discussion and give criticisms of the code or procedures. Sometimes this criticism revolves around the use of free versus non-free tools and other times it is related to attempts to fix bugs in the code. In either case, the GNUe maintainers who discuss these critiques respect and respond to outsiders reviews with serious consideration even without knowing the reviewer's credentials.

6 GNUe Case Study

The GNUe case study consists of the analysis of three cases of software development communication over the IRC. They involve 1) the debate over the use of a non-free tool for creation of a graphic; 2) the debate over the use of a non-free tool for GNUe documentation creation and maintenance; and 3) the initiation of a newcomer who fixes bugs in realtime. Each case will be described briefly in this section. For a more detailed description, see (Elliott and Scacchi, 2003).

6.1 Case One – Use of Non-Free Graphic Tool for Documentation

In this section we present the first case study that reveals a trajectory of a conflict and debate over the use of a non-free tool to create a graphic on the GNUe Web site (See http://www.gnuenterprise.org/irc-logs/gnue-public.log.25Nov2001). This exchange takes place on November 25, 2001 on the IRC channel and ends the next morning. This example illustrates the ease with which a newcomer comes onboard and criticizes the methods used to produce a graphical representation of a screenshot on the GNUe Web site. CyrilB, an outsider to GNUe, finds a graphic that was created using Adobe Photoshop, a non-free graphical tool. He begins the interchange with a challenge to anyone onboard stating that "it is quite shocking" to see the use of non-free software on a free software project. He exhibits a **strong belief in free software,** which causes a debate lasting a couple of days. Table 1 displays the total number of contributors and the number of days of the conflict. Eight of the nine regular GNUe contributors were software developers and one was working on documentation. The infrequent contributors drifted on and off throughout the day – sometimes lurking and other times involved in the discussion.

Total Contributors	Regular Contributors	Infrequent Contributors	Number of Days
17	9	8	1

Table 1 – Contributors and Duration of Conflict in Case One

The **strong belief in free software** of the outsider leads to conflict among those insiders who have a moderate view of the use of free software for GNUe software development. A daylong debate ensues among the Neilt, creator of the graphic, CyrilB, and other GNUe contributors regarding the use of a non-free software tool to create a graphic for a GNUe screenshot for Web site documentation.

CyrilB uses his **strong view of belief in free software** to promote the spirit of the free software movement by exclaiming that images on the gnuenterpise.org Web site seem to be made with non-free Adobe software. His reaction provokes strong reactions from GNUe contributors:

> "I hope I'm wrong: it is quite shocking…We should avoid using non-free software at all cost, am I wrong? **(Strong BIFS-1)**"

Reinhard responds with a **moderate view of belief in free software:**

> "Our main goal is to produce good free software. We accept contributions without regarding what tools were used to do the work especially we accept documentation in nearly any form we can get because we are desparate for documentation." **(Moderate View BIFS-1).**

Once CyrilB has pointed out the use of the non-free graphic, Neilt, who originally created the GNUe diagram using Adobe Photoshop, joins the IRC, reviews the previous discussion on the archived IRC, and returns to discuss the issue with Reinhard and CyrilB. A lively argument ensues between Neilt and others with onlookers contributing suggestions for the use of free tools to develop the Adobe graphic.

Meanwhile Maniac, who has been "listening" to this debate, jumps in and gives technical details about a PNG image. Then Reinhard and Neilt agree that CyrilB had a valid point since a PNG has no vector information stored and so it would be difficult to use free software to edit the graphic. These exchanges illustrate how participants use the IRC medium to support and enable the cooperative work needed to resolve this issue. It also conveys the community spirit and cooperative work ethic that is a value in the GNUe work culture. They both agree to wait until CyrilB comes back to give more suggestions for an alternative.

Outside critiques of software and procedures used during development are common to the GNUe project. One of the norms of the work culture is **immediate acceptance of outsider**

contributions. Eventually, Neilt, the creator of the non-free graphic questioned CyrilB's qualifications and was satisfied when he learned that CyrilB was a member of the European Free Software Foundation. However, he was willing to fix the graphic prior to the revelation of CyrilB's credentials.

Consequences of the debate are a reinforcement of the **belief in free software, value in community,** and **value in cooperative work;** and a recreation of a Web site graphic with free software to replace the original created with a non-free software tool.

6.2 Case Two – Use of Non-Free Software for GNUe Documentation

The second case study explores project insider review of the procedures and practices for developing GNUe documentation (See http://www.gnuenterprise.org/irc-logs/gnue-public.log.15Nov2001 for the full three day logs). Once again the debate revolves around polarized views of the use of non-free tools to develop GNUe documentation. In this case, Chillywilly, a frequent contributor, balks at the need to implement a non-free tool on his computer in order to edit the documentation associated with a current release. Even though his colleagues attempt to dissuade him from his concerns by suggesting that he can use any editor – free or non-free- to read the documentation in HTML or other formats, Chillywilly refuses to back down from his stance based on a **strong belief in free software**. This debate lasts three days. Table 2 displays the number of contributors and their classification for participation in case two. This case exemplifies the fierce adherence to the belief in free software held by some purists in the free software movement and how it directs the work of the day. While the three day debate reinforces beliefs and values of the culture, at the same time, it ties up valuable time which could have been spent writing code or documentation, yet it contributes to community building.

Total Contributors	Regular Contributors	Infrequent Contributors	Number of Days
24	9	15	3

Table 2 – Contributors and Duration of Conflict over Documentation

In order to understand this example, some background information is needed. The GNUe core maintainers selected a free tool to use for all documentation called *docbook* (http://www.docbook.org). DocBook is based on an SGML document type definition which provides a system for writing structured documents using SGML or XML. However, several GNUe developers as of November 15, 2001 were having trouble with its installation. Consequently, they resorted to using lyx tool to create documentation (http://www.lyx.org)...

The problem with lyx is that even though it was developed as a free software tool, its graphical user interface (GUI) requires the installation of a non-free graphics package (called *libxforms*). Chillywilly gets upset with the fact that he has to install non-free software in order to read and edit GNUe documentation. A lengthy discussion ensues with debates over which tool to use for GNUe documentation. This debate lasts for three days taking up much of the IRC time until Chillywilly finally gives up the argument. The strength in the **belief in free software** drives this discussion. The debate and its resolution also illustrate the tremendous effort by developers to collaborate and work cooperatively through the use of the IRC channel. Although the discussion is heated at moments, a sense of fun also pervades. Chillywilly begins on the November 14, 2001 IRC with an observation that a fellow collaborator, jamest, has made documents with lyx:

> Action: chillywilly trout whips jamest for making lyx docs
> Action: jcater troutslaps chillywilly for troutslapping jamest for making easy to do docs
> <chillywilly> lyx requires non-free software
> <Maniac> lyx rules
> <chillywilly> should that be acceptable for a GNU project?

<jcater> chillywilly: basically, given the time frame we are in, it's either LyX documentation with this release, or no documentation for a while (until we can get some other stinking system in place)
<jcater> pick one :)
<chillywilly> use docbook then

…

<Maniac> lyx's graphics library is non-gpl **(i.e. non-free software)**
<chillywilly> I'm not writing your docs for you
<Maniac> this is an issue the developers are aware of but do not, at this time, have the time to rectify
<chillywilly> Maniac: because they are **** KDE nazis
<chillywilly> that's who the original lyz authors are matthias, et. al.
<Maniac> well, my understanding is, they are working toward UI independance, to make it able to use differnt toolkits ie. kde, gnome, xyz as time/coding permit

Maniac questions chillywilly's incessant reminders about using non-free software as though this myopic view of free software development is unnecessary. Chillywilly continues his debate showing his **strong view of free software**.

Reinhard agrees with chillywilly as do others, but in order to complete the documentation, they agree to use an interim solution. Chillywilly is so adamantly opposed to the use of non-free software that he references Richard Stallman as part of his reasoning – "**I will NOT install lyx and make vrms unhappy**". This passage shows how RMS is considered the "guru" of the free software movement. Eventually chillywilly sends an email to the mailing list:

> **"**OK, I saw on the commit list that you guys made some LyX documents. I think it is extremely ***that a GNU project would require me to install non-free software in order to read and modify the documentation. I mean if I cannot make vrms happy on my debian system them what good am I as a Free Software developer? Is docbook really this much of a pain? I can build html versions of stuff on my box if this is what we have to do. This just irks me beyond anything. I really shouldn't have to be harping on this issue for a GNU project, but some ppl like to take convenience over freedom and this should not be tolerated… Is it really that unreasonable to request that we not use something that requires ppl to install non-free software? Please let me know. (Chillywilly, mailing list)"

A lengthy discussion of technical issues unrelated to the documentation problem ensues. Meanwhile Jcater has sent a reply to Chillywilly's message to the mailing list.:

> "I would like to personally apologize to the discussion list for the childish email you recently received. It stemmed from a conversation in IRC that quickly got out of hand. It was never our intention to alienate users by using a non-standard documentation format such as LyX. Writing documentation is a tedious chore few programmers enjoy. The developers of the GNUe client tools are no exception…The upcoming release was originally planned for this past weekend. James and I decided to postpone the release… LyX was chosen because it is usable and, more importantly, installable. After many failed attempts at installing the requirements for docbook, James and I made the decision that LyX-based documentation with the upcoming 0.1.0 releases was better than no documentation at all…
> PPS, By the way, Daniel, using/writing Free software is NOT about making RMS happy or unhappy. He's a great guy and all, but not the center of the free universe, nor the motivating factor in many (most?) of our lives. For me, my motivation to be here is a free future for my son (Jcater, mailing list)."

The belief in freedom is a motivating factor for Jcater as stated above, even freedom for his son.

6.3 Case Three – Newcomer Asking for Help with GNUe Installation

In this example, mcb30 joins the IRC as a newcomer who wants to install and use GNUe business applications for his small business in England (http://www.gnuenterprise.org/irc-logs/gnue-public.log.16Nov2001). In addition, he offers his services as a contributor and immediately starts fixing bugs in realtime. This case is a good example of the community building spirit of GNUe since mcb30 is immediately accepted by frequent contributors especially because he posts significant bug fixes very rapidly.

> <mcb30> Is anyone here awake and listening?
> <reinhard> yes
> <mcb30> Excellent. I'm trying to get a CVS copy of GNUe up and running for the first(ish) time - do you mind if I ask for a few hints?
> <reinhard> shoot away :)
> <reinhard> btw what exactly are you trying to run?
> <reinhard> as "GNUe" as a whole doesen't exist (yet)
> <reinhard> GNUe is a meta-project (a group of related projects)
> <mcb30> OK - what I want to do is get *something* running so I can get a feel for what there is, what state of development it's in etc. - I'd like to contribute but I need to know what already exists first!
> <reinhard> ok cool
> <reinhard> let me give you a quick overview
> <mcb30> I have finally (about 5 minutes ago) managed to get "setup.py devel" to work

properly - there are 2 bugs in it
<mcb30> ok

Mcb30 goes offline and continues to fix bugs. He then comes back and suggests that he has a patch file to help

<mcb30> I've got a patch file - who should I send it to? jcater?
<reinhard> jcater or jamest
<mcb30> ok, will do, thanks
<reinhard> mcb30: btw sorry if i tell you things you already know :)
<mcb30> don't worry - I'd rather be told twice than not at all! :-)
<reinhard> people appearing here in IRC sometimes have _very_ different levels of information :)

<reinhard> look at examples/python/addrbook.py
<mcb30> excellent, thanks!
<mcb30> will have a play around
<reinhard> mcb30: i will have to thank you
<reinhard> mcb30: we are happy if you are going to help us
<reinhard> gotta leave now

Later mcb30 comes back to the IRC and posts code that he wrote to fix a problem and several frequent contributors thank him and say that they wish they could hire him for pay. As with the first case, contributors immediately accept them into the "club" and, as the chat unfolds, they ask him for his credentials, motivation, and location (mcb30 is an educational consultant for IT in the English school system).

7 Discussion

The three examples from the GNUe case study will be discussed in this section in relation to the three main themes found in the data: realtime teamwork, building community, and conflict resoluton. Each example comes from a detailed conding and content analysis of the IRCs.

7.1 Building Community

Kollock (1996) suggests that there are design principles for building a successful online community such as identity persistence. He draws upon the work of Godwin (1994) showing

that allowing users to resolve their own disputes without outside interference and providing institutional memory are two principles for making vritual communities work. Applying these principles to the GNUe project shows that disputes are resolved simultaneously via IRC, and recorded in IRC archives as a form of institutional memory. In the GNUe virtual community, the community is continuously changing (when newcomers join even if for a brief time) yet the core maintainers are dedicated for long periods of time. Here is a quote from Derek, a core maintainer, who believes that the IRC helps them sustain their community:

> "Many free software folks think IRC is a waste of time as there is 'goofing off', but honestly I can say its what builds a community. I think a community is necessary to survive. For example GNUe has been around for more than 3 years. I can not tell you how many projects have come and gone that were supposed be competition or such. I put our longevity solely to the fact that we have a community." (Derek, email interview (2002))

7.2 Conflict Resolution

In the two conflict resolution GNUe cases presented here, both issues resulted in a solution by debate on the IRC and mailing lists. In the first case, the contributor who created the graphic with ADOBE photoshop agreed to change it in the future using a free tool. In the second case, chillywilly stopped badgering his co-workers about the use of a non-free graphics package to complete documentation. His colleagues essentially told him to get back to work and use a text editor if he is so worried about the use of *lyx* until they all can use the free software *docbook*. In both cases, the conflicts were resolved in a reasonable amount of time via the IRC exchanges. At the same time, the beliefs in free software are reinforced by people defending their positions and this, in turn, helps to perpetuate the community.

7.3 Facilitating Teamwork

In each case there was evidence that as the day proceeded on the IRC, people were going offline to experiment with free software that would help to resolve the conflict (i.e. a free graphics

package and a free text editor). Many infrequent contributors or newcomers who were lurking and watching the problem unfold on the IRC, also gave technical advice for a tool to use to solve the problem. The realtime aspect of the work clearly facilitates the teamwork since people could simultaneously work together solving a technical problem. In the third case, a newcomer who was having trouble with the GNUe installation was directed by a maintainer to the original author of the code. Surprisingly, the original author joins the IRC that day and discusses the bugs with mcb30.

8 Practical Implications

We have shown that the persistent recording of daily work using instant messaging (IRC) and Kernel Cousins can serve as a community building avenue. Managers of open source might benefit from incorporating these CMC mediums into their computing infrastructures. It assists employees in conflict management and also binds the groups together by reinforcing the organizational culture. As illustrated in the non-conflict GNUe example, the IRC serves as an expertise Q&A repository. The author of the software quickly emerged and mcb30 was able to gain detailed knowledge of how the system works. In addition, the IRC enables realtime software design and debugging. As F/OOSD projects proliferate, managers should consider the benefits of using an IRC to facilitate software development and to help build a community..

9 Future Research

We plan to continue with the analysis of GNUe data and compare the results with other free software communities. Likewise, we expect to find similar beliefs and values in some open source projects and plan to explore this phenomena. In this way, we can assertain whether GNUe is in fact a unique culture (Martin, 2002) or whether other free software projects have similar software development processes. A review of other GNU projects shows evidence of

proselitization of beliefs in free software (http://www.gnu.org/projects/projects.html). In addition, in the LINUX community, there is an ongoing dispute about using Bitkeeper (non-free) versus CVS (free) as a case management system. Other future research of interest is to determine if having strong beliefs and values regarding free software contribute to a successful, productive F/OOSD community.

10 Conclusions

Previous CSCW research has not addressed how the collection of IRC messaging, IRC transcript logs, and email lists, and periodic digests (Kernel Cousins) can be collectively mobilized and routinely used to create a virtual organization that embodies, transmits, and reaffirms the cultural beliefs, values, and norms such as those found in free software projects like GNUe. Strong organizational cultural beliefs in an F/OOSD virtual community combined with persisent recordation of chat logs tie a group together and helps to build a community and perpetuate the project. The beliefs in freedom, free software, and freedom of choice create a special bond for the people working on free software projects. These beliefs foster the values of cooperative work and community-building. Schein's (1990) theory of organizational culture includes revelation of underlying assumptions of cultural members that are on a mostly unconscious level. In the GNUe world, the underlying assumptions of cooperative work and community-building become ingrained in the everyday work practices in their pursuit of an electronic business and ERP system implemented as free software. These beliefs and values enhance and motivate acceptance of outsiders' criticisms and resolution of conflict despite the distance separation and amorphous state of the contributor population.

11 References

Avison, D. E., & Myers, M. D. (1995). Information Systems and Anthropology: An Anthropological Perspective on IT and Organizational Culture. *Information Technology and People, 8*, (43-56).

Berquist, M. and J. Ljungberg, The power of gifts: organizing social relationships in open source communities, *Info. Systems J.*, 11(4), 305-320, October 2001.

Crowston, K., & Scozzi, B. (2002). *Exploring Strengths and Limits on Open Source Software Engineering Processes: A Research Agenda.* Paper presented at the *2nd Workshop on Open Source Software Engineering*, Orlando, Florida.

DiBona, C., Ockman, S., & Stone, M. (1999). *Open Sources: Voices from the Open Source Revolution*. Sebastol, CA: O'Reilly & Associates Inc.

Dubé, L., & Robey, D. (1999). Software Stories: Three Cultural Perspectives on the Organizational Practices of Software Development. *Accounting, Management and Information Technologies, 9*(4), 223-259.

Easterbrook, S. (Ed.). (1993). *CSCW: Cooperation or Conflict*. New York: Springer-Verlag.

Elliott, M. (2000). *Organizational Culture and Computer-Supported Cooperative Work in a Common Information Space: Case Processing in the Criminal Courts*. (Vol. Unpublished Dissertation). Irvine: University of California, Irvine.

Elliott, M. (2003). *The Virtual Organizational Culture of a Free Software Development Community.* Paper presented at the *3rd Workshop on Open Source Software*, Portland, Oregon.

Elliott, M., & Scacchi, W. (2002). Communicating and Mitigating Conflict in Open Source Software Development Projects, Working Paper: Institute for Software Research, University of California, Irvine, http://www.ics.uci.edu/~melliott/commossd.htm

Elliott, M., & Scacchi, W. (2003). Free Software: A Case Study of Software Development in a Virtual Organizational Culture. Working Paper: Institute for Software Research, University of Calfornia Irvine, http://www.ics.uci.edu/~wscacchi/Papers/New/Elliott-Scacchi-GNUe-study-DRAFT.pdf

Feller, J., & Fitzgerald, B. (2002). *Understanding Open Source Software Development*. N.Y.: Addison-Wesley.

Fielding, R. T. (1999). Shared Leadership in the Apache Project. *Communications of the ACM, 42*(4), 42-43.

K. Fogel (1999). *Supporting Open Source Development with CVS.* Scottsdale, AZ: Coriolis Press.

Easterbrook, S. M., Beck, E. E., Goodlet, J. S., Plowman, M., Sharples, M., & Wood, C. C. (1993). A Survey of Empirical Studies of Conflict. In S. M. Easterbrook (Ed.), *CSCW: Cooperation or Conflict?,* 1-68. London: Springer-Verlag.

Godwin, M. (1984). Nine Principles for Making Virtual Communities Work. *Wired,* 2.06, 72-73.

Gregory, K. (1983). Native-view Paradigms: Multiple Cultures and Culture Conflicts in Organizations. *Administrative Science Quarterly, 28*, 359-376.

James D. Herbsleb, Rebecca E. Grinter (1999). Splitting the Organization and Integrating the Code: Conway's Law Revisited. ICSE 1999: 85-95.

Hine, C. (2000). *Virtual Ethnography*. London: Sage.

Koch, S., & Schneider, G. (2000). *Results from Software Engineering Research into Open Source Development Projects Using Public Data*, Wirtschaftsuniversitat Wien.

Kollock, P. (1996). Design Principles for Online Communities, *The Internet and Society: Harvard Conference Proceedings,* http://sscnet.ucla.edu/soc/faculty/kollock/papers/design.htm.

Kollock, P., & Smith, M. (1996). Managing the Virtual Commons: Cooperation and Conflict in Computer Communities. In S. Herring (Ed.), *Computer-Mediated Communication: Linguistic, Social, and Cross-Cultural Perspectives*, 109-128, Amsterdam: John Benjamins.

Kollock, P., & Smith, M. A. (1999). Communities in Cyberspace. In M. A. Smith & P. Kollock (Eds.), *Communities in Cyberspace* (pp. 3-25). New York, NY: Routledge.

Mackenzie, A., Rouchy, P., & Rouncefield, M. (2002). *Rebel Code? The Open Source 'Code' of Work.* Paper presented at the Open Source Software Development Workshop, February 25-26, 2002, Newcastle-upon-Tyne, UK.

Martin, J. (2002). *Organizational Culture: Mapping the Terrain*. Thousand Oaks: Sage Publications.

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

Mockus, A., Fielding, R. T., & Herbsleb, J. (2000). A Case Study of Open Source Software Development: The Apache Server. *Proc. 22nd Intern. Conf. on Software Engineering*, 263-272, Limerick, IR.

Nardi, B., Whittaker, S., Bradner, E. Interaction and Outeraction: Instant Messaging in Action. (2000). Proceedings CSCW 2000.

Noll, J., & Scacchi, W. (1999). Supporting Software Development in Virtual Enterprises. *Journal of Digital Information,* 1(4), http://jodi.ecs.soton.ac.uk/.

Olsson, S. (2000). *Ethnography and Internet: Differences in Doing Ethnography in Real and Virtual Environments.* Paper presented at the IRIS 23, Laboratorium for Interaction Technology, University of Trollhattan Uddevalla.

Ott, J. (1989). *The Organizational Culture Perspective*. Pacific Grove, CA: Brooks/Cole.

Pace, R. C. (1990). Personalized and Depersonalized Conflict in Small Group Discussions: An Examination of Differentiation. *Small Group Research, 21*(1), 79-96.

Pavlicek, R. G. (2000). *Embracing Insanity: Open Source Software Development*. Indianapolis, IN: SAMS Publishing.

Raymond, E. S. (2001). *The Cathedral & The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA: O'Reilly & Associates.

Robey, D., & Azevedo, A. (1994). Cultural Analysis of the Organizational Consequences of Information Technology. *Accounting, Management, and Information Technology,* 4(1), 23-37.

Sawyer, S. (2001). Effects of Intra-Group Conflict on Packaged Software Development Team Performance. *Information Systems Journal, 11*, (155-178).

Scacchi, W. (2002a). *Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development* (Technical Report). Irvine, CA: University of California, Irvine.

Scacchi, W. (2002b). Understanding Requirements for Developing Open Source Software Systems. *IEE Proceedings - Software, 149*(2), 24-39.

Schein, E. (1990). Organizational Culture. *American Psychologist, 45*, 109-119.

Schein, E. H. (1991). The Role of the Founder in the Creation of Organizational Culture. In P. J. Frost, L. F. Moore, M. R. Louis, C. C. Lundberg, & J. Martin (Eds.), *Reframing Organizational Culture* (pp. 14-25). Newbury Park, CA: SAGE Publications, Inc.

Schein, E. H. (1992). *Organizational Culture and Leadership*. San Francisco: Jossy-Bass.

Smith, A. D. (1999). Problems of Conflict Management in Virtual Communities. In M. A. Smith & P. Kollock (Eds.), *Communities in Cyberspace* (pp. 134-163). New York, NY: Routledge.

Stallman, R. (1999a). *Free Software Foundation Brochure*. Cambridge, MA: Free Software Foundation.

Stallman, R. (1999b). The GNU Operating System and the Free Software Movement. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open Sources: Voices from the Open Source Revolution* (pp. 53-70). Sebastopol, CA: O'Reilly & Associates, Inc.

Strauss, A. L., & Corbin, J. (1990). *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Newbury Park, CA: Sage Publications.

Trice, H. M., & Beyer, J. M. (1993). *The Cultures of Work Organizations*. Englewood Cliffs, NJ: Prentice Hall.

Williams, S. (2002). *Free as in Freedom: Richard Stallman's Crusade for Free Software*. Sebastopol, CA: O'Reilly & Associates.

Yin, R. K. (1994). *Case Study Research, Design and Methods.* 2nd ed. Newbury Park: Sage Publications.

Collaboration, Leadership, Control, and Conflict Negotiation in the *Netbeans.org* Open Source Software Development Community

Chris Jensen and Walt Scacchi
Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA, USA 92697-3425
{cjensen, wscacchi}@ics.uci.edu

Abstract

Large open source software development communities are quickly learning that, to be successful, they must integrate efforts not only among the organizations investing developers within the community and unaffiliated volunteer contributors, but also negotiate relationships with external groups hoping to sway the social and technical direction of the community and its products. Leadership and control sharing across organizations and individuals in and between communities are common sources of conflict. Such conflict often leads to breakdowns in collaboration. This paper seeks to explore the negotiation of these conflicts, collaborative efforts, and leadership and control structures in the Netbeans.org community.

Keywords

Collaboration, Conflict Negotiation, Leadership, Process, Open Source Software Development, Netbeans.org

1. Introduction

Is open source software development (OSSD) best characterized as being strictly cooperative, or as cooperative and in conflict at the same time [Easterbrook 1993]? Conflict clearly arises during sustained software development efforts [e.g., Sawyer 2001]. But previous studies of conflict associated with Internet-based communities has focused attention to that found in specific OSSD projects operating as virtual organizations [Elliott and Scacchi 2003], as non-profit foundations [O'Mahony 2004], or in online discussion communities [Smith 1999]. None of these studies specifically help us understand the kinds of conflict, cooperation, and collaboration that arises or is needed to coordinate large-scale OSSD processes and effort in large project communities where corporate sponsorship may be a central facet of OSSD.

NetBeans.org is one of the largest OSSD communities around these days [cf. Jensen and Scacchi 2003]. Netbeans.org is a Java-focused OSSD community backed by Sun Microsystems devoted to creating both an integrated development environment (IDE) for developing large Java-based applications, as well as a platform for development of other software products. Originally started as a student project in 1996, the Netbeans.org project was acquired and subsequently released as an open source community project by Sun, whose Netbeans.org team includes many of the community's core developers. While the issues presented here stem from observations in the Netbeans.org community, they are by no means limited to this community, nor have their challenges been insurmountable.

Our study focuses on three items. First, we identify the objects of interaction among participants in the NetBeans.org community that are media through which collaboration, leadership, control and conflict negotiation are expressed and enacted. Second, we explore relationships arising in NetBeans.org on an intra-community level. Then, we look at relationships between communities like Netbeans.org and other communities and organizations.

2. Objects of Interaction

Much of the development work that occurs in an open source software project centers around the creation, update, and other actions (e.g., copy, move, delete) applied to a variety of software development artifacts. These artifacts serve as coordination mechanisms [Schmidt and Simone 1996, Simone and Mark 1999], in that they help participants communicate, document, and otherwise make sense of what the emerging

software system is suppose to do, how it should be or was accomplished, who did what, what went wrong before, how to fix it, and so forth. Furthermore, within a project community these artifacts help coordinate local, project-specific development activities, whereas between multiple project communities, these artifacts emerge as boundary objects [Star 1990] through which inter-community activities and relations are negotiated and revised. The artifacts may take the form of text messages posted to a project discussion list, Web pages, source code directories and files, site maps, and more, and they are employed as the primary media through which software requirements and design are expressed. These "software informalisms" [Scacchi 2002] are especially important as coordination mechanisms in OSSD projects since participants generally are not co-located, they do not meet face-to-face, and authority and expertise relationships among participants is up for grabs.

The NetBeans IDE is intended to support the development of Web-compatible Java applications. In the context of the NetBeans.org project and its role within a larger Web-compatible information infrastructure, additional artifacts come into play within and across projects. These include the content transfer protocols like the HyperText Transfer Protocol (http) which are systematically specified in Internet standards like RFC documents, as well as more narrowly focused communication state controllers associated with remote procedure calls (or remote method invocations). They also include shared data description formats like the HyperText Markup Language (html) and the eXtensible Markup Language (XML), as well as client-side or server-side data processing scripts (e.g., CGI routines). Such descriptions may be further interpreted to enable externally developed modules to serve as application/module plug-ins, which enable secondary or embedded applications to be associated with an OSS system. Other artifacts are brought in from other OSSD projects to serve as project support tools, such as those used to record and store system defect (bug) reports (Issuzilla), email list managers, and even large comprehensive collaborative software development environments and project portals, like SourceCast [Augustin, Bressler, and Smith 2002]. Finally, OSSD projects may share both static and operational artifacts in the course of collaborating or cooperating through mutually intelligible and interoperable development processes, which might take an explicit form like the Java Community Process (JCP), or an implicit and embedded form such as that which emerges from use of project repositories whose contents are shared and synchronized through tools that control and track alternative versions (CVS), bug reports, or Web site content updates.

Accordingly, in order to explore where issues of collaboration, leadership, control and conflict may arise within or across related OSSD projects, then one place to look to see such issues is in how project participants create, update, exchange, debate, and make sense of the software informalisms that are employed to coordinate their development activities. This is the approach taken here in exploring the issues both within the NetBeans.org project community, as well as across the (fr)agile ecosystem [Highsmith 2002] of inter-related OSSD projects that situate NetBeans.org within a Web information infrastructure.

3. Intra-Community Issues

As noted in the first section, NetBeans.org is a large and complex OSSD project. To help convey a sense of the complexity, semi-structured modeling techniques such as rich pictures [Monk and Howard 1998] can be used to provide a visual overview of the context that situates the creation and manipulation of the software informalisms and OSSD processes that can be observed in the NetBeans.org project [Oza, *et al*, 2002]. Figure 1 displays such a rich picture, highlighting the variety of roles that participants in the NetBeans.org project perform, the types of concerns they have in each role, and the development tasks they regularly enact, as part of the configuration of OSSD activities they articulate and coordinate through software informalisms [cf . Simone and Mark 1999].

We have observed at least three kinds of issues arise *within* an OSSD community like NetBeans.org. These are collaboration, leadership and control, and conflict.

3.1. Collaboration

According to the Netbeans.org community Web site, interested individuals may participate in the community by joining in discussions on mailing lists, filing bug and enhancement reports, contributing Web content, source code, newsletter articles, and language translations. These activities can be done in isolation, without coordinating with other community members, and then offered up for consideration and inclusion. As we'll see, reducing the need for collaboration is a common practice in the community that gives rise to positive and

negative effects. We discuss collaboration in terms of policies that support process structures that prevent conflict, looking at task completion guidelines and community architecture.

3.1.1. Policies and Guidelines

The NetBeans.org community has detailed procedural guidelines[1] for most common development tasks, from submitting bug fixes to user interface design and creating a new release. These guidelines come in two flavors: development task and design style guidelines. In general, these policies are practiced and followed without question. Ironically, the procedures for policy revision have not been specified.

Precedent states that revisions are brought up on the community or module discussion mailing lists, where they are debated and either ratified or rejected by consensus. Developers are expected to take notice of the decision and act accordingly, while the requisite guideline documents are updated to reflect the changes. In addition, as some communities resort to "public flogging" for failure to follow stated procedures, requests for revision are rare and usually well known among concerned parties, so no such flogging is done within Netbeans.org.

Overall, these policies allow individual developers to work independently within a process structure that enables collaboration by encouraging or reinforcing developers to work in ways that are expected by their fellow community members, as well as congruent with the community process.

3.1.2. Separation of Concerns: an Architectural Strategy for Collaborative Success

Software products are increasingly developing a modular, plug-in application program interface (API) architectural style in order to facilitate development of add-on components that extend system functionality. This strategy has been essential in an open source arena that carries freedom of extensibility as a basic privilege or, in some cases, the right of free speech or freedom of expression through contributed source code. But this separation of concerns strategy for code management also provides a degree of separation of concerns in developer management, and therefore, collaboration.

In concept, a module team can take the plug-in API specification and develop a modular extension for the system using any development process in complete isolation from the rest of the community. This ability is very attractive to third-party contributors in the Netbeans.org community who may be uninterested in becoming involved with the technical and socio-political issues of the community, or who are unwilling or unable to contribute their source code back to the community. Thus, this separation of concerns in the Netbeans.org design architecture engenders separation of concerns in the process architecture. Of course, this is limited by the extent that each module in the Netbeans.org community is dependent on other modules.

Last, volunteer community members have periodically observed difficulties collaborating with volunteer community members. For example, at one point a lack of responsiveness of the (primarily Sun employed) user interface team[2], whose influence spans the entire community, could be observed. This coordination breakdown led to the monumental failure of usability efforts for a period when usability was arguably the most-cited reason users chose competing tools over Netbeans.org. Thus, a collaboration failure gave rise to product failure. Only by overcoming collaboration issues was Netbeans.org able to deliver a satisfactory usability experience[3].

3.2. Leadership and Control

Ignoring internal Sun (and third party) enterprise structure, there are five observable layers of the Netbeans.org community hierarchy. Members may take on multiple roles some of which span several of these layers. At the bottom layer are users, followed by source contributors, module-level managers, project level release managers (i.e. IDE *or* platform), and finally, community level managers (i.e. IDE *and* platform) at the top-most layer. Interestingly, the "management" positions are simply limited to coordinating roles; they carry no other technical or managerial authority. The release manager, for example, has no authority to determine what will be included in and excluded from the release[4]. Nor does s/he have the authority to assign people to complete the tasks required to release the product. The same is true of module and community

1 http://www.netbeans.org.org/community/guidelines/

2 http://www.netbeans.org.org/servlets/ReadMsg?msgId=531512&listName=nbdiscuss

3 http://www.javalobby.org/thread.jspa?forumID=61&threadID=9550#top

4 http://www.netbeans.org.org/community/guidelines/process.html

managers. Instead, their role is to announce the tasks that need to be done and wait for volunteers to accept responsibility.

Accountability and expectations of responsibility are based solely on precedent and volunteerism rather than explicit assignment, leading to confusion of the role of parties contributing to development. Leadership is not asserted until a community member champions a cause and while volunteerism is expected, this expectation is not always obvious. The lack of a clear authority structure is both a cause of freedom and chaos in open source development. Though often seen as one of its strengths in comparison to closed source efforts, it can lead to process failure if no one steps forward to perform critical activities or if misidentified expectations cause dissent.

The difficulties in collaboration across organizations within the community occasionally brought up in the community mailing lists stem from the lack of a shared understanding leadership in the community. This manifests itself in two ways: a lack of transparency in the decision making process and decision making without community consent. While not new phenomenon, they are especially poignant in a movement whose basic tenets include freedom and knowledge sharing.

3.2.1. Transparency in the Decision Making Process

In communities with a corporately backed core development effort, there are often decisions made that create a community-wide impact that are made company meetings. However, these decisions may not be explicitly communicated to the rest of the community. Likewise private communication between parties that is not made available on the community Web space or to the forwarded to other members is also hidden. This lack of transparency in decision-making process makes it difficult for other community members to understand and comply with the changes taking place if they are not questioned or rejected. This effect surfaced in the Netbeans.org community recently following a discussion of modifying the release process [cf. Erenkrantz 2003][5].

Given the magnitude of contributions from the primary benefactor, other developers were unsure of the responsibility and authority Sun assumed within the development process. The lack of a clearly stated policy outlining these bounds led to a flurry of excitement when Sun members announced major changes to the licensing scheme used by the community without any warning. It has also caused occasional collaboration breakdown throughout the community due to expectations of who would carry out which development tasks. The otherwise implicit nature of Sun's contributions in relation to other organizations and individuals has been revealed primarily through precedent rather than assertion.

3.2.2. Consent in the Decision Making Process

Without an authority structure, all decisions in development are done through consensus, except among those lacking transparency. In the case of the licensing scheme change, some developers felt that Sun was within its rights as the major contributor and the most exposed to legal threat [6] while others saw it as an attack on the "democratic protection mechanisms" of the community that ensure fairness between participating parties[7]. A lack of consideration and transparency in the decision making process tend to alienate those who are not consulted and erode the sense of community.

3.3. Conflict Resolution

Conflicts in the Netbeans.org community are resolved via community discussion mailing lists. The process usually begins when one member announces dissatisfaction with an issue in development. Those who also feel concern with the particular issue then write responses to the charges raised. At some point, the conversation dissipates- usually when emotions are set aside and clarifications have been made that provide an understanding of the issue at hand. If the problem persists, the community governance board is tasked with the responsibility of resolving the matter.

The governance board is composed of three individuals and has the role of ensuring the fairness throughout the community by solving persistent disputes. Two of the members are elected by the community, and one is appointed by Sun Microsystems. The board is, historically, a largely superficial entity whose authority and scope are questionable and untested. While it has been suggested that the board intercede on a few rare

5 http://www.netbeans.org/servlets/BrowseList?listName=nbdiscuss&by=thread&from=19116&to=19116&first=1&count=41

6 http://www.netbeans.org.org/servlets/ReadMsg?msgId=534707&listName=nbdiscuss

7 http://www.netbeans.org.org/servlets/ReadMsg?msgId=534520&listName=nbdiscuss

occasions, the disputes have dissolved before the board has acted. Nevertheless, board elections are dutifully held every six months[8].

Board members are typically prominent members in the community. Their status carries somewhat more weight in community policy discussions, however, even when one member has suggested a decision, as no three board members have ever voted in resolution on any issue, and thus, it is unclear what effect would result. Their role, then, is more of a mediator: to drive community members to resolve the issue amongst themselves. To this end, they have been effective.

4. Inter-Community Issues

As noted earlier, the NetBeans.org project is not an isolated OSSD project. Instead, the NetBeans IDE which is the focus of development activities in the NetBeans.org project community is envisioned to support the interactive development of Web-compatible software applications or services that can be accessed, executed, or served through other OSS systems like the Mozilla Web browser and Apache Web server. Thus, it is reasonable to explore how the NetBeans.org project community is situated within an ecosystem of inter-related OSSD projects that facilitate or constrain the intended usage of the NetBeans IDE. Figure 2 provides a rendering of some of the more visible OSSD projects that surround and embed the NetBeans.org within a Web information infrastructure. This rendering also suggests that issues of like collaboration and conflict can arise at the boundaries between projects, and thus these issues constitute relations that can emerge between project communities in OSSD ecosystem.

With such a framing in mind, we have observed at least three kinds of issues arise *across* OSSD communities that surround the NetBeans.org community. These are communication and collaboration, leadership and control, and conflict resolution.

4.1. Communication and Collaboration

In addition to their IDE, Netbeans.org also releases a general application development platform on which the IDE is based. Other organizations, such as BioBeans and RefactorIT communities build tools on top of or extending the NetBeans platform or IDE. How do these organizations interact with Netbeans.org, and how does Netbeans.org interact with other IDE and platform producing organizations? For some organizations, this collaboration may occur in terms of bug reports and feature requests submitted to the Netbeans.org issue-tracking repository. Additionally, they may also submit patches or participate in discussions on community mailing list or participate in the Netbeans.org "Move the Needle" branding initiative. Beyond this, Netbeans.org participates in the Sun sponsored *Java.net* meta-community, which hosts hundreds of Java-based OSSD projects developed by tens of thousands of individuals and organizations.

A fellow member of the Java.net community, the Java Tools Community, considered by some to be a working group[9] for the Java Community Process, is an attempt to bring tool developers together to form standards for tool interoperability. Thus Netbeans.org, through its relationship with Sun, is a collaborating community in the development of, and through compliance with, these standards, and looks to increasing collaboration with other tool developing organizations.

4.2. Leadership and Control

OSSD generally embrace the notion of choice between software products to build or use. At the same time, developers in any community seek success for their community, which translates to market share.

In some cases, communities developing alternative tools do so in peaceful coexistence, even collaboratively. In other cases, there is a greater sense of competition between rivals. NetBeans and its chief competitor Eclipse (backed largely by IBM) fall into the latter category. Eclipse has enjoyed some favor from users due to performance and usability issues of NetBeans, as well as IBM's significant marketing and development resource contributions. Yet, they have a willingness to consider collaborative efforts to satisfy demands for a single, unified IDE for the Java language that would serve as a platform for building Java development tools and a formidable competitor to Microsoft's .NET. Ultimately, the union was defeated, largely due to technical and organizational differences between Sun and IBM[10], including the inability or

8 http://www.netbeans.org.org/about/os/who-board.html

9 http://www.internetnews.com/dev-news/article.php/3295991

10 http://www.adtmag.com/article.asp?id=8634, and

unwillingness to determine how to integrate the architectures and code bases for their respective user interface development frameworks (Swing for NetBeans and SWT for Eclipse).

4.3. Conflict Resolution

Conflicts between collaborating communities are resolved in similar fashion to their means of communication- through discussion between Sun and Eclipse representatives, comments on the Netbeans.org mailing lists, or other prominent technical forums (e.g. Slashdot and developer blogs). Unfortunately, many of these discussions occur after the collaborating developer has moved away from using Netbeans.org (often, in favor of Eclipse). Nevertheless, the feedback they provide gives both parties an opportunity to increase understanding and assists the Netbeans.org community by guiding their technical direction.

5. Discussion and Conclusion

Generally, volunteer Netbeans.org developers expect Sun to provide leadership but not control. People outside the community (e.g. users, former users, and potential users) often voice their concerns in off-community forums (e.g., Slashdot, blogs, etc) rather than NetBeans.org community message boards, due to accountability or visibility barriers (creating an account, logging in accounts), small as they may seem to be. In addition, such message forums may not be a part of such an individual's daily work habits- they're more likely to visit a site like Slashdot.org than the Netbeans.org forum because they are not interested enough in staying abreast of NetBeans developments or participating in the community. Nonetheless, people working in, or interested in joining or studying OSSD projects, must address how best to communicate and collaborate their development processes and effort, how to facilitate or ignore project leadership and control, and how to work you way through conflicts that may or may not be resolvable by community participants.

Overall, we have observed three kinds of coordination and collaborating issues arise within OSSD project communities like NetBeans.org, and three similar kinds of issues arise across OSSD communities that surround NetBeans.org within an ecosystem of projects that constitute a Web information infrastructure. Previous studies of conflict in either OSSD projects have examined either smaller projects, or in virtual communities that do not per se develop software as their focus. As corporate interest and sponsorship of OSSD stimulates the formation of large projects, or else the consolidation of many smaller OSSD projects into some sort of for-profit or not-for-profit corporate enterprise for large-scale OSSD, then we will need to better understand issues of collaboration, conflict, and control in OSSD.

http://www.eweek.com/article2/0,1759,1460110,00.asp

6. Acknowledgments

The research described in this report is supported by grants from the National Science Foundation #ITR-0083075 and #ITR-0205679 and #ITR-0205724. No endorsement implied. Contributors to work described in this paper include Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; and Margaret Elliott at the UCI Institute for Software Research.

7. References

Augustin, L., Bressler, D., and Smith, G., Accelerating Software Development through Collaboration, *Proc. 24th Intern. Conf. Software Engineering*, Orlando, FL, 559-563, May 2002,

Crowston, K. and Scozzi, B., Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings—Software*, 149(1), 3017, 2002.

Easterbrook, S. (ed.), *CSCW: Cooperation or Conflict*, Springer-Verlag, New York, 1993.

Elliott, M. The Virtual Organizational Culture of a Free Software Development Community, *Proc. 3rd Workshop on Open Source Software Engineering, 25th. Intern. Conf. Software Engineering*, Portland, OR, May 2003.

Elliott, M. and Scacchi, W., 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 2003.

Erenkrantz, J., Release Management within Open Source Projects, *Proc. 3rd Workshop on Open Source Software Engineering, 25th Intern. Conf. Software Engineering*, Portland, OR, May 2003.

Highsmith, J., *Agile Software Development Ecosystems,* Addison-Wesley Pub. Co., 2002.

Jensen, C. and Scacchi, W., Automating the Discovery and Modeling of Open Source Software Processes, *Proc. 3rd Workshop on Open Source Software Engineering*, 25th. Intern. Conf. Software Engineering, Portland, OR, May 2003.

Jensen, C. and Scacchi, W., Process Modeling of the Web Internet Infrastructure, submitted for publication, June 2004.

Monk, A. and Howard, S. The Rich Picture: A Tool for Reasoning about Work Context, *Interactions*, 21-30, March-April 1998.

O'Mahony, S., Non-profit Foundations and their Role in Community-Firm Software Collaboration, to appear in *Making Sense of the Bazaar: Perspectives on Open Source and Free Softwar*e, J. Feller, B. Fitzgerald, S. Hissam, & K. Lakhani (Eds.), O'Reilly & Associates, Sebastopol, CA, 2004.

Sawyer, S., Effects of intra-group conflict on packaged software development team performance, *Information Systems J.,* 11, 155-178, 2001.

Scacchi, W., Understanding the Requirements for Developing Open Source Software, *IEE Proceedings—Software*, 149(1), 24-39, 2002.

Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004.

Schmidt, K., and Simone, C., Coordination Mechnanisms: Towards a Conceptual Foundation of CSCW System Design, *Computer Supported Cooperative Work,* 5(2-3), 155-200, 1996.

Simone, C. and Mark, G., Interoperability as a Means of Articulation Work, *Proc. Intern. Joint Conf. Work Activities Coordination and Collaboration,* San Francisco, CA, 39-48, ACM Press, 1999.

Smith, A.D., Problems of Conflict Management in Virtual Communities. In M.A. Smith and P. Kollock (eds.), *Communities in Cyberspace,* Routledge, New York, 134-163, 1999.

Star, S. L., The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving, in *Distributed Artificial Intelligence* (eds. L. Gasser and M. N. Huhns), Vol. 2, 37-54. Pitman, London, 1990.

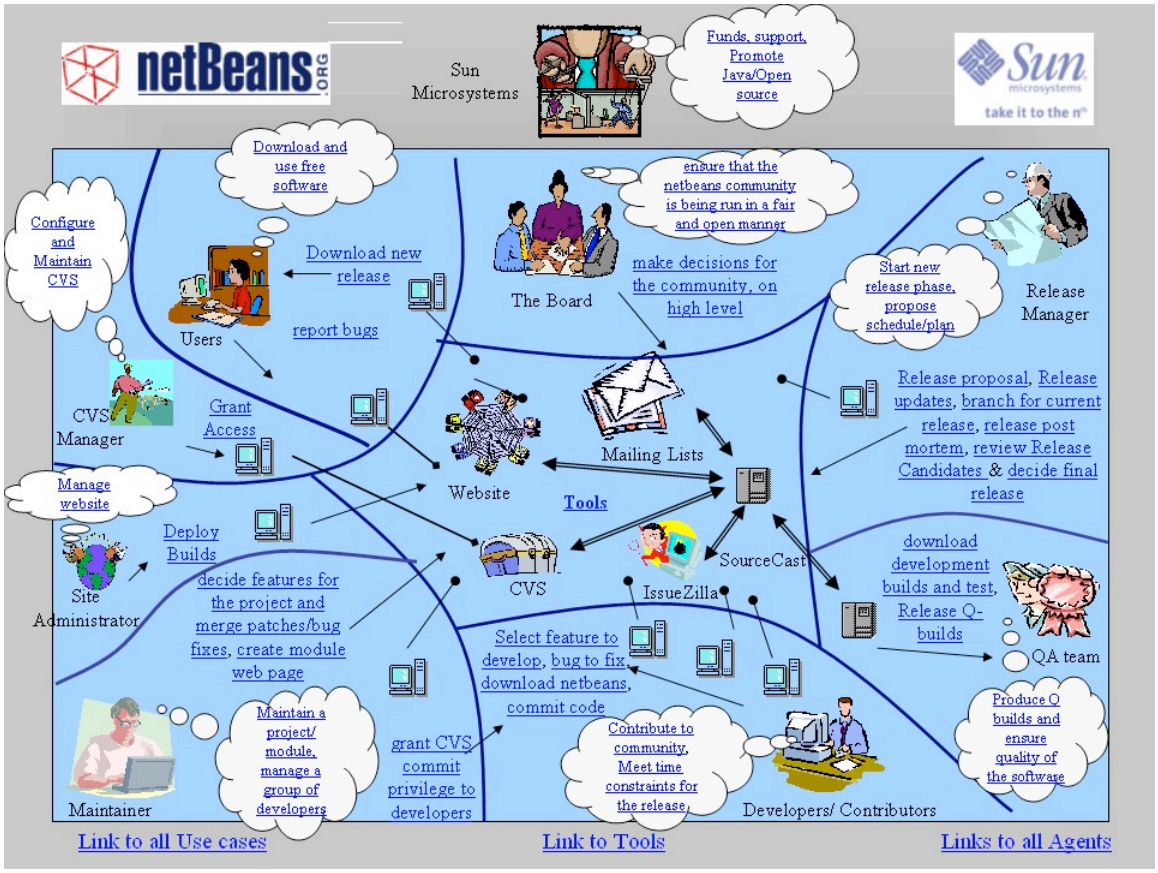


Figure 1. A rich picture view of the roles (labeled icons), concerns (clouds), and activities (hyperlinked text) found in the intra-community context of NetBeans.org [cf. Oza, *et al*, 2002].

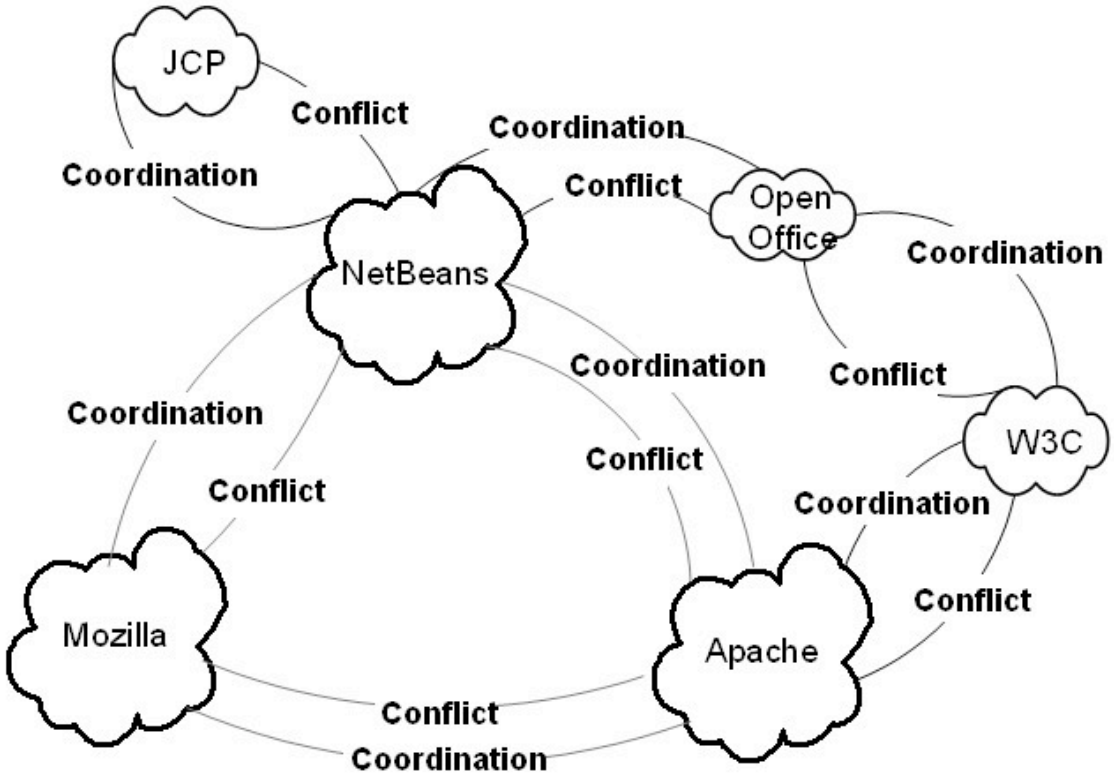


Figure 2. An overview of the inter-community ecosystem for NetBeans.org

Mobilization of Software Developers: The Free Software Movement

Margaret S. Elliott
Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697
949-824-8756
melliott@ics.uci.edu

Walt Scacchi
Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697
949-824-4130
wscacchi@ics.uci.edu

August 2004

Revised version to appear in *Information, Technology, and People,*
Festschrift for Rob Kling

ABSTRACT
Free software and open source software development projects are growing at a rapid rate. Globally dispersed virtual communities with large groups of software developers contribute time and effort often without pay. One force behind this phenomenon is the Free Software Movement (FSM), a 20 year-old social movement whose purpose is to promote the use of free software instead of proprietary software. We show how the ideology of the FSM influences software development work practices in free software development communities. We show how the belief in free software, and the values of cooperative work and community building, are manifested in FSM norms of copyleft software licensing, team-based informal management with fluctuating membership, and open communication. We also give examples from a case study of a free software development community, GNU Enterprise. We discuss the differences between free software development and typical in-house development with suggestions for what businesses can learn from free software development.

Keywords
Free software movement, computerization movements, organizational culture, free software development, open source software, virtual community, internetworking

INTRODUCTION

Free and open source software development projects are growing at a rapid rate. The SourceForge Web site reports 850,000+ users with 800 new ones joining every day, and a total of 85,000+ projects with 80 new ones added each day. Globally dispersed virtual communities (Kollock and Smith, 1999) have formed with large groups of software developers freely contributing time and effort to free and open source software projects. Most people are familiar with the widespread use of Linux distributions, such as Debian GNU/Linux, which combines more than 100 million lines of source code, representing a Unix-like operating system kernel, with nearly 10,000 software utilities, tools, and applications.[i] However, there are many smaller-sized free or open source projects in various stages of software development that are revolutionalizing the way software systems are designed, implemented, and maintained.

A strong force behind this emergent phenomenon is the Free Software Movement (FSM), a 20 year-old computerization movement whose purpose is to promote the development and use of free software instead of proprietary software.[ii] The FSM and its ideological center, the Free Software Foundation (FSF), support the mission to preserve, protect and promote the freedom to use, study, copy, modify, and redistribute computer software, and to defend the rights of free software users to exercise or experience such freedom. The FSF's website states that the "ultimate goal is to provide free software to do all of the jobs computer users want to do—and thus make proprietary software obsolete." The FSM also admonishes the use of non-free software as immoral because it prevents its users from learning (about computer programming from the study of source code) and prevents developers and users from helping their fellow man (Stallman, 2002; Williams, 2002).

In this paper, we examine the FSM with respect to the ideology that characterizes it and that is promoted by the FSF, and how such ideals give rise to beliefs, values, and norms that affect technical choices and work practices in free software development. We present the framing of this vision as a virtual organizational culture with the belief in free software, and the values in cooperative work and building community manifested in norms including the copyleft[1] licensing technique; informal management with fluctuating membership; and open communication regarding design, code, test, and release of software. We present research results from a case study of a free software community to support the framing of the FSM ideology.

The open source movement is a related movement advocated by the Open Source Initiative[iii] started in 1998 as an alternative to the FSM. The term "open source " grew out of a strategy meeting in February 1998 of a group of key "free software" advocates who believed that a new label and approach was needed to attract businesses to the concept of using free software. Today, many practicing software developers aligned with these movements, use the terms "free software" and "open source" interchangeably. However, the research community recognizes differences between the meanings of these terms. The research literature often refers to "free/open source software" (F/OSS) to refer

[1] *Copyleft* refers to the type of license recommended by the FSF to be used for the copyright of free software which basically means that if you distribute free software, you must also grant the right to the user to have the freedom to read, modify, and redistribute the source code.

to both types of software together. However, the FSF clearly distinguishes itself from open source stating that "Open source is a development methodology; free software is a social movement."[iv]

Another way to view the relationship between free and open source software is that free software qualifies as open source software according to the open source initiative[v], but open source does not always qualify as free software according to the FSF definition of free software.[vi] Both movements advocate specific software copyright requirements as part of their respective definitions,[vii] so that these copyright and usage licenses are also carriers of ideological differences in free and open source software movements (Scacchi 2004). It is beyond the scope of this paper to outline in detail the differences between these two movements. Instead, we focus on the FSM as a computerization movement with its ideology to avoid non-free software.

Rob Kling and Suzanne Iacono coined the term "computerization movement" as a type of social movement whose activists promote mass computerization of specific computing technologies to bring about a new social order. (Kling and Iacono, 1988; Iacono and Kling, 1996). They argue (Iacono and Kling, 2001) that the meaning of the Internet is built up or "framed" in macro-level discourses such as those of the government, media and scientific disciplines. People who participate in computerization movements build up technological action frames[2] in their public discourses that specify positive links between internetworking[3] and a new, preferred social order without regard to the social costs or organizational changes needed for this societal change. Furthermore, Iacono and Kling (2001) point out that there is a lack of studies that focus on the practices of distributed work with networked computing systems within or across organizations – e.g. how people communicate, how leadership works, and what the impacts of the work are on the people who engage in them. Subsequently, we build on this research through examination of the ideological components that frame FSM discourse in situated practices and their influence on distributed work within free software projects.

We show how the ideological frames of the FSM in the form of beliefs and values result in normative changes to software development practices within free software projects through the public discourse of books and articles published by the FSF, and technical

[2] Frames are cognitive structures or mental models that are held by individuals and are considered to be shared structure when there is a significant overlap of cognitive categories and content. Technological frames are conceptual or analytic lenses that form within and between social groups as they struggle over the meaning of a technology and form it as part of their discourses (Orlikowski and Gash, 1994). Iacono and Kling (2001) used the term "technological action frames" to characterize frames that shape and structure public discourse regarding the use of a particular technology, and that legitimate high levels of investment for potential users, and form the core ideas about how a technology works and how a future based on its use should be envisioned.

[3] The Internet is typically defined as a network of networks, or an internetwork. Internetworking means using special-purpose computers to connect a variety of autonomous networks for video, video or graphic formats over distances (Iacono and Kling, 2001). The webopedia online dictionary at http://www.webopedia.com/TERM/I/internetworking.html defines Internetworking as: The art and science of connecting individual local-area networks (LANs) to create wide-area networks (WANs) , and connecting WANs to form even larger WANs. Internetworking can be extremely complex because it generally involves connecting networks that use different protocols. Internetworking is accomplished with routers, bridges, and gateways. Other forms: *internetwork* (v.)

artifacts such as websites, mailing list archives, Internet Relay Chat (IRC) archives, and summary digest archives. These frames help to maintain the boundary that distinguishes free software from open source software as well as from proprietary software practices. In previous papers, we presented the results of a qualitative study of the methods and social processes used in GNU enterprise[viii] (GNUe), a free software development community whose purpose is to build a free enterprise resource planning (ERP)[4]. We characterized the GNUe free software community as a virtual organizational culture (Martin, 2002; Schein, 1992; Trice and Beyer, 1993) showing how beliefs and values are manifested in three cases of GNUe software development (Elliott, 2003; Elliott and Scacchi, 2003a) – two cases of conflict and one of cooperative work. We presented an analysis of cases of cooperation and conflict in GNUe IRC sessions (Elliott and Scacchi, 2004). We also characterized the FSM as an occupational community (Elliott and Scacchi, 2003b). In addition, we showed how they jointly build community and software system artifacts through a web of socio-technical interactions and relations (Kling and Scacchi, 1982; Scacchi, 2002a). In this paper, we show how beliefs and values of the FSM are manifested in the norms of free software development communities. We compare the work practices of a typical free software development project with those of a typical in-house software project. Lastly, we make suggestions for what we can learn from free software development that could enhance in-house software development.

COMPUTERIZATION MOVEMENTS

Social movements can be defined as "collective enterprises to establish a new order of life" (Blumer, 1969:8). The groups that form around a computer technology form a social movement with mobilizing ideologies that promote an improved social order or oppose a bad one; form organizations with a variety of membership; and promote the social movement via various communication modes and patterns of computing resource deployment. Kling and Iacono (1988) identified five specific computing technologies: urban information systems, artificial intelligence, computer-based education, office automation, and personal computing. These movements help shape public images of computers and computerization as technological progress while deflecting competing social values. Kling and Iacono (1988) found the following commonalities among these movements:

1) Computer-based technologies are central for a reformed world.
2) The improvement of computer-based technologies will help reform society.
3) No one loses from computerization.
4) More computing is better than less, and there are no conceptual limits to the scope of appropriate computerization.
5) Perverse or undisciplined people are the main barriers to social reform through computing.

More recently, Iacono and Kling (2001), used the analytical framework of technological action frames which shape public discourse to show how such frames influence organizational practices through five recurring beliefs that inform the master frame

[4] ERP systems are large packaged software systems that are employed in corporate and government enterprises to manage and track resources through the semi-automated production and operational processes, and to schedule the replacement of consumable resources used by these processes (Skok and Legge 2002).

around internetworking. They are nearly identical to the five listed in (Kling and Iacono, 1988) except for the one stating that "Internetworking pushes the conceptual limits of time, space, and the known world." They argue that computerization movement activists imply that there are no limits to meaningful internetworking by downplaying the actual limits of new technologies, while accenting the continuing benefits of current technologies, as though the Internet were the only place for meaningful activities to occur (Iacono and Kling, 2001). Building from here, we examine the FSM ideology as informing a new type of computerization movement, where instead of proselytizing the benefits of the Internet, FSM activists assume that internetworking is a given, and that it is accessible to all who are recruited to contribute to the growth and perpetuation of free software projects.

FREE SOFTWARE MOVEMENT IDEOLOGY

The FSM community shares the following similar characteristics with other online communities: 1) working together or communicating at a distance; 2) providing open access to all members; 3) providing archives of mailing lists; 4) building a community; and 5) bonding of members to help sustain community as a group with similar beliefs, values and/or interests. However, the fact that the FSM is a work-centered endeavor with a focus on changing society's view of software development, places the FSM in a unique category of computerization movements with these differences:

1) The cost to organizations to adopt "the best" computer technologies to fulfill computerization movement visions can be quite high (e.g., a manager might purchase PCs for employees who might benefit from a movement's technology). In this era of ubiquitous computing, an assumption is made that participants are not hampered from joining the FSM due to lack of resources.

2) Most computerization movements are supported and promoted by mobilizing organizations and/or advocates in public press. Although the FSM is promoted by the FSF, Richard Stallman has been the key activist responsible for its flourishing membership over the last 20 years. He has since been joined by fellow believers in the U.S.-based FSF, as well as its international counterparts, the European FSF and Indian FSF. However, many people respect and revere Richard Stallman as the founder and creative guru of the FSM. He is considered a noteworthy visionary and has received several monetary awards to support his cause, including a prestigious MacArthur Foundation "genius" award, and the 2001 Takeda Award for techno-entrepreneurial achievement for social/economic well-being in the amount of $830,000 (Williams, 2002).

In the next section, we describe our two research sites: the FSF and GNU Enterprise followed by a section on Research Methods.

RESEARCH SITE: THE FREE SOFTWARE FOUNDATION AND THE GNUE ENTERPRISE COMMUNITY

We selected the FSF as one of the research sites due to its central role in establishing and perpetuating the FSM. The FSF is a non-profit organization that supports the FSM mission to preserve, protect and promote the freedom to use, study, copy, modify, and redistribute computer software, and to defend the rights of free software users. Their operating costs were funded last year mainly by individual donations (67%). The GNU Project, a precursor to the FSF, was formed by Richard Stallman in 1984 to develop a complete UNIX-style operating system as free software: the GNU system. (GNU is a recursive acronym for "GNU's Not UNIX"). This system has evolved into the GNU/Linux system using the Linux kernel combined with GNU utilities.

We selected the GNU Enterprise (GNUe)[ix] virtual community as our research site to study processes of free software development and the work practices of free software developers (Elliott, 2003; Elliott and Scacchi, 2003a; 2003b; 2004; Scacchi, 2002b). GNUe is a meta-project of the GNU[x] Project that exists on the Web, but otherwise has no physical place of operation or business. GNUe seeks to develop: 1) set of tools that provide a development framework for information technology professionals to create or customize applications and share them across organizations; 2) a set of packages written using the set of tools to implement a full ERP system; and 3) a general community of support and resources for developers writing applications using GNUe Tools.

GNUe operates as an international virtual organization for software development based in the U.S. and Europe. It is centered about the GNUe Web portal and global Internet infrastructure that enables remote access and collaboration. As many as twelve companies located across the U.S. and Europe sponsor paid GNUe participants. These companies provide salaried personnel, computing resources, and infrastructure that support this organization. However, many project participants support their participation through other means.

RESEARCH METHODS

In this section, we describe the types of data and analysis performed on each. See Appendix A for a table of data and corresponding methods and interview protocols.The sources of data for this virtual ethnography (Hine, 2000) include books and articles on free and open source software development; instant messaging (Nardi *et al*., 2000; Herbsleb and Grinter, 1999); transcripts captured through IRC logs; threaded email discussion messages; and other Web-based artifacts associated with the FSF and the GNU project such as Kernel Cousins (summary digests of the IRC and mailing lists)[xi]. The use of a Web site by the FSF to archive articles and information about the GNU project and the philosophy regarding free software provided a rich background for this study. In addition, the GNUe project records design decisions, detailed documentation of code, virtual meetings on IRC logs, and summary digests on their Web site. We interpreted project artifacts, books, and documents, as well as Web site descriptions of free and open source software processes (Scacchi, 2002a; b). We discovered strong cultural overtones in the readings and began searching for a site to apply an analysis of how motivations and

cultural beliefs influenced the social processes of free software development (Elliott, 2002). The FSF and GNUe Web sites offered public access to downloadable IRC archives and mailing lists as well as lengthy documentation - all facilitating a virtual ethnography (Hine, 2000).

This research also includes data from email and face-to-face interviews with GNUe contributors, and observations at Open Source conferences. The first author spent over 200 hours studying and perusing IRC archives and mailing list samples during open and axial coding, and analysis phases of the grounded theory (Strauss and Corbin, 1990). During the open coding phase, the first case study concerning conflict over tool choice, presented in (Elliott and Scacchi, 2002a), was selected as representative of the strong influence of cultural beliefs on GNUe software development practices. We took each GNUe IRC and kernel cousin related to three cases (two concerning conflict over ideology and coding practices and one concerning cooperative work), and applied codes derived from the data (Strauss and Corbin, 1990). In this way, we discovered relationships between the codes derived in the open coding phase.

During the axial coding phase of several IRC chat logs, mailing lists and other documentation, we discovered relationships between beliefs and values of the work culture and manifestations of the culture. For a detailed presentation of the variables and relationships, see (Elliott and Scacchi, 2003a). We discovered that for some GNUe participants, their strong belief in the development and use of free software was an idealistic motivation for joining and perpetuating the community. In addition, we explored the influence of the FSM on the ideology of GNUe and its work practices. We then began characterizing free and open source software developers as an occupational community with a quest for self-control of affairs and with varying strengths in beliefs and values (Elliott and Scacchi, 2003). Last, we focus our attention in this paper on the inception and growth of the FSM as a computerization movement and on its influence on the work practices of sites such as GNUe. See Appendix A for more details on Research Methods.

FREE SOFTWARE MOVEMENT BELIEFS

Beliefs form the core of ideologies, and as such, are an important component of any cultural study, as well as a driving force that empowers a social movement like the FSM. The FSF promotes the belief in free software by posting articles on its Web site, distributing brochures, publishing books, by giving speeches at computer conferences, and by directing people to where to find free software on the Web. In our previous research on the GNUe community (Elliott and Scacchi, 2003a, 2003b, 2004; Scacchi, 2002b), we showed how many GNUe programmers were motivated to participate by their beliefs in free software. We also discovered other beliefs such as freedom of choice in what to work on or when.

Recent studies of the motivations of individuals for contributing to free/open source software development coincide with our findings regarding the ideological motivation of free software contributors. Intrinsic motivations such as pleasure in programming and identity with the open source community were primary reasons for people to participate in

free/open source programming (Hars and Ou, 2002; Lakhani and Wolf, 2005). In the survey of 79 free/open source programmers (Hars and Ou, 2002), 16.5 percent of respondents rated high on altruism and 30 percent identified strongly with the open-source community or had a kin-like relationship with other open-source programmers. In a larger survey of 684 free/open source contributors from 287 projects (Lakhani et. al (2004), 42% strongly agreed and 41% somewhat agreed that the "hacker"[5] community is their primary source of identity, while 30 percent believed that all source code should be open. Similar to our GNUe case study results where interviewees expressed the importance of "giving back to the free software community", 28.6 percent surveyed were motivated by giving code back to the free/open source community. Here we discuss the belief in free software as central to the FSM.

Belief in Free Software

Belief in freedom is a recurring theme throughout the data. In fact, the free software movement is claimed by RMS to be inspired from the ideals of the American Revolution of 1776: freedom, community, and voluntary cooperation (Stallman, 2001a). During a lengthy heated debate regarding the use of free versus non-free software to develop documentation, one GNUe developer claimed that he is working on GNUe for the "freedom of his son" (GNUe Mailing List archive, 2001). The GPL was designed by RMS for the following reason:

> "to uphold and defend the freedoms that define free software – to use the words of 1776, it establishes them as inalienable rights for programs released under the GPL. It ensures that you have the freedom to study, change, and redistribute the program, by saying that nobody is authorized to take these freedoms away from you by redistributing the program under a restrictive license" (Stallman, 2001a).

The following is an excerpt from an essay titled "The Free Software Definition" found on the FSF Website explaining the concept of free software:

> "We maintain this free software definition to show clearly what must be true about a particular software program for it to be considered free software. "Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech", not as in "free beer". Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:
>
> - The freedom to *run* the program, for any purpose (freedom 0).
> - The freedom to *study* how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.

[5] A slang term for a computer enthusiast, i.e., a person who enjoys learning programming languages and computer systems and can often be considered an expert on the subject(s). Among professional programmers, depending on how it used, the term can be either complimentary or derogatory, although it is developing an increasingly derogatory connotation. The pejorative sense of *hacker* is becoming more prominent largely because the popular press has coopted the term to refer to individuals who gain unauthorized access to computer systems for the purpose of stealing and corrupting data. Hackers, themselves, maintain that the proper term for such individuals is cracker.

- The freedom to *redistribute* copies so you can help your neighbor (freedom 2).
- The freedom to *improve* the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this."

> The belief in free software is a core motivator of free software developers. GNUe developers show a strong belief in free software extolling its virtues on its Web site and in daily activity on the IRC logs. This belief is manifested in electronic artifacts such as the Web pages, source code, software design diagrams, and accompanying articles. The GNUe Web site advertises that it is "a free software project with a corps of volunteer developers around the world working on GNUe projects". The GNUe Web site home page [xii] advertises itself as a world-wide enterprise. The Web site provides a link to an article by RMS, as well as declaring that its purpose is "putting the free back into free enterprise." The GNUe software is licensed under the GNU GPL[xiii].

Throughout the publicly available GNUe summary digests and IRC logs found on the Web, there are numerous references to the importance of adhering to the principles of free software. The next section describes the values of the FSM.

FREE SOFTWARE MOVEMENT VALUES

The core values identified for the free software culture are building community and cooperative work. These values were evident in our study of free software Web sites, documentation, and informative articles and books (DiBono *et al*., 1999; Pavlicek, 2000; Raymond, 2001). Here we present how these values are promoted by the FSM and manifested in the GNUe community.

Building Community

Articles on the FSF Web site emphasize the importance of cooperating on free software projects and in contributing to the free software community. The GNUe online community exists for the purpose of developing a free ERP system. In some cases we found evidence of building community in the GNUe organizaiton as stated below:

> "Many free software folks think IRC is a waste of time as there is 'goofing off', but honestly I can say its what builds a community. I think a community is necessary to survive. For example, GNUe has been around for more than 3 years. I can not tell you how many projects have come and gone that were supposed be competition or such. I put our longevity solely to the fact that we have a community." (Derek, email interview (2002))

We also found evidence of building community in the IRC archives when newcomers join GNUe offering contributions and existing contributors quickly accept them as part of the community (See (Elliott and Scacchi, 2002) for details).

Cooperative Work

The FSM community's belief in free software and value in community foster a value in cooperative work. The FSF Web site is replete with proselitization for FSM followers to cooperate with each other and to create as much free sofware for the world as possible. Throughout the GNUe IRC archives, there is evidence of cooperative work when conflict arises and many contributors and lurkers try to resolve the technical problem, and when new people join GNUe, others are eager to welcome them, answer questions, and give advise, etc. (See (Elliott and Scacchi, 2003a; b) for a detailed account). As part of this cooperative work practice, members of GNU projects sometimes offer detailed suggestions to future free software programmers. Havoc Pennington, author of the book, GTK+/Gnome Application Development, and former member of the GNOME Foundation board of directors (GNOME is a free desktop graphical interface for Linux), offers advice on how to start and maintain a free software project in an article titled "Working on Free Software." [xiv]

. A summary of the typcial steps listed in his article follow:

4) Establish the need for the program by studying the free projects that are already in existence.
5) Once the need for a new project is established, start coding until enough is completed for a release to the general public. At this point, one needs to register on savannah.org that a new project is in the works with a core maintainer actively working on the code and in need of free software developers and hope for volunteers to come forth. Once a project is established, people usually use a software configuration management system like CVS to track changes.
6) Some unexpected circumstances include:
    a. No one is in charge. The maintainer should not expect to control deadlines or assignments.
    b. Coordination with others is essential so efforts are not duplicated.
    c. Be wary of back-seat coders who suggest brilliant ideas for modules or changes to code but who never code or do not know how to code.
    d. Selecting the appropriate software license and using it properly is important. The FSF suggests the use of the GPL or GPL-like licenses..
7) Learn about the community and its culture remembering that people are volunteers and must be treated with respect.

In the next section, we discuss the norms representing the manifestation of the FSM beliefs and values.

NORMS

We address the norms that we believe are most important from the FSM community: copyleft, informal management with fluctuating membership, and open communication.

Copyleft

"Copyleft" is a general method for copyrighting and licensing a computer program as free software with the requirement that all modifications and extensions to the software are free as well. One way to license a program is to place it in the public domain as an uncopyrighted piece of software for others to copy, modify, and potentially distribute as a

proprietary program. The FSF instead recommends that free software developers use the GNU GPL or one of its derivatives[xv]. By including a GPL-type license with every free software program, developers spread the philosophy and freedom intended by Richard Stallman when he founded the FSF. In this way, Richard Stallman was able to ensure that the freedoms that he so fervently advocates are bound legally to the code. In his opinion, proprietary software developers use copyright to take away users' freedom, whereas the FSF uses copyright to guarantee their freedom.

Informal Management with Fluctuating Membership

Free software development represents a relatively new approach to the development of complex software systems (Feller and Fitzgerald, 2002). In most situations, the resulting software system and its associated Web-based documents or development artifacts are globally accessible at little or no direct cost. One of the key issues in the success of a free software project is it ability to manage software development without a top manager monitoring activity and passing judgment on the quality and timeliness of the work (Scacchi, 2002a; Scacchi, 2004).
There is no lead organization or prime contractor that brings the alliance of individuals and sponsoring firms as a network virtual organization. It is more of an emergent organizational form where participants have in a sense discovered each other, and have brought together their individual competencies and contributions in a way whereby they can be integrated or made to interoperate (Crowston and Scozzi, 2002; Ljungberg, 2000). Membership flucutates everyday from lurkers to unpaid regulars to full-time employees (rare but occasionally paid). In addition, the work is accomplished in a setting where many people communicate anonymously and *never* meet face-to-face to organize or manage the software contributions offered to the free software project. However, free and open source software projects have a set of one or more core maintainers (often the owner(s) of the original idea behind the software project) who decide(s) what design decisions to make about modifications to the code base, what to put in a release, which bugs to fix, etc.

In our GNUe case study, no company has administrative authority or resource control to determine: (a) what work will be done; (b) what the schedule will be; (c) who will be assigned to perform specified tasks; (d) whether available resources for the project are adequate; nor (e) who will be fired or reassigned for inadequate job performance (Scacchi 2002b). The project is monitored by a small group of core maintainers who are responsible for a large bulk of the code, test, and release of software. Most of these people monitor and interact with others on the IRC during some time period every day. There is also a list of frequent contributors who work on software development, documentation, and other administrative duties. GNUe also welcomes a crew of casual volunteers (those who contribute once or twice, or who work sporadically). The participants come from different small companies or act as individuals that collectively act to move the GNUe software and the GNUe community forward. Thus, the participants self-organize in a manner more like a meritocracy (Fielding, 1999; Scacchi, 2004).

A core maintainer explains the typical method of managing the GNUe software assignments as:

> "The number one rule in free software is 'never do timelines or roadmaps'. This is a problem in open source projects. We could use a better roadmap, not having one hinders us. The features we add come about by need during consulting implementations. We may need some kind of roadmap in the future as we expand with more people." (Derek, face-to-face interview, August 2002)

Although GNUe contributors are not subject to a timeline, the group of core maintainers on GNUe make decisions about module design, what bugs get fixed in what release, and whether or not to accept a contributor's design idea or piece of code (for new software or a bug fix). In that sense, the random group of GNUe contributors has some semblance of orderly managerial control over what code actually comprises the final product of GNUe. In a typical organization, computer systems can become an occasion for increasing social control over selected behaviors (Kling and Iacono, 1984) by revealing institutional data about their work captured in a computer repository or report. In this setting, social control is more of a persuasive nature in which core maintainers might gently chastise someone over the IRC delays in completing a piece of code, code documentation, or design document. A convivial spirit pervades these exchanges since the expectations of core maintainers is not based on managerial control (contributors are not being paid by maintainers to do the job so accountability for completed work is different than in a typical in-house software project).

In two of the GNUe case studies (Elliott and Scacchi, 2003), we found evidence of debates over the use of non-free versus free tools. In both cases, the group resolved the conflict without a top manager needing to intervene. Both examples illustrate how the strong beliefs in free software by two contributors can result in a heated discussion on the IRC and mailing list archives (Elliott and Scacchi, 2003; 2004).

Researchers have studied how technology embodies structures (built in by designers during development), which are then appropriated by users during their implementation and adaptation of the technology to fit their needs (Orlikowski, 2000). The recurrent use of the same type of technology across various contexts and practices results in different technologies-in-practice referring to the specific structure routinely enacted when one uses a particular technology or computer program in recurrent ways in everyday situated actions (Orlikowski, 2000). The GNUe core maintainers have designed their free software to enact technologies-in-practice into their open design of the software enabling companies to add or modify modules to the ERP system as they see fit. In addition, by using the GPL license and by their association with other free software projects, they are encouraging users to potentially add their situated context to the total free package by contributing their modules back to the GNUe Website downloadable releases.

Open Communication

The FSM and FSF promote the open communication of all design and code transactions. Since source and executable files associated with a free program are generally available for downloading from a Web site, all communication among core maintainers, programmers, documentation writers, etc. becomes archived in a central public location on the Web site. When new "free" software projects are initiated, they are added to the

Web site list calling attention to the need for volunteers. The FSF offers their computers as a site for setting up a central place for downloads and upgrades of free software releases.

For that reason, the Free Software Foundation encourages GNU software projects to use the machines at gnu.org as a home base. Using these machines also benefits the GNU Project indirectly, by increasing public awareness of GNU, and spreading the idea of working together for the benefit of everyone."[xvi]

These software artifacts take the form of mailing list archives, IRC archives, code and design documentation, HOWTOs, FAQs, summary digests, user guides, and community Wikis. In this way, the public discourse regarding the belief in free software is perpetuated by constant reinforcement of beliefs in this public organizational memory (Ackerman and Halverson, 2000).

The discussions conducted for design, work and "play" occur during any part of the day since the work of free software projects includes contributors across global time zones. Results of requirements, design, code, test, and release "meetings" are posted on mailing list archives, chat archives, summary digests, and code documentation. In the GNUe world, the favorite communication medium of core contributors appears to be the IRC. In the next section, we discuss how these methods differ from typical in-house software development.

FREE SOFTWARE WORK PRACTICES VERSUS INHOUSE WORK PRACTICES

Table 1 summarizes differences between free software practices and closed source typically used in software development firms. Combining free and/or open source software with proprietary software is becoming more acceptable even on aerospace mission critical software (Norris, 2003), due to its cost savings measures and increasingly reliable results (Scacchi, 2002a). Here we attempt to clarify the major differences between free and non-free software development practices along six dimensions listed in column one. These dimensions represent an expanded view of the norms listed in the previous section on the FSM Norms.

Table 1 - Differences between Free and Closed Software Development

Category	**Free software development norms**	**Non-free software development norms**
Digital Property Rights	Copyleft – GPL or similar with GPL-based freedoms	Copyright of software with no derivative works or freedoms
Publication of source code and related documentation	Online publication of all source code, code documentation, communication docs, email archives, etc.	Closed publication of company's source code, documentation, memos, etc.
Management of software production	Team-based "loose" management of schedules and due dates (no boss, and no predetermined schedule of assignments)	Hierarchical management with "top dog(s)" of company monitoring project status; managers accountable for schedule due dates
Acceptance of contributors to project	Programmers welcomed without credential check; constantly changing membership with many people finding and fixing bugs.	Programmers hired after reference checks, interviews, merit-based pay scale, steady employees, one or two people testing and fixing bugs.
Compensation for work	Programmers may work as hobbyists, some work for pay, some work for reputation or to learn more	Programmers work for pay, and won't work if not paid.
Communication to fellow workers regarding work	Open communication for general public to review, GNUe uses IRC and other free software media: summary digests, mailing lists.	Closed communication (or open in-house but closed to outside world)

SUMMARY AND CONCLUSIONS

We have shown how the FSM has influenced the evolution of free software development communities with the belief in the freedoms of free software and the values in cooperative work and building community. We discussed how the FSM ideology promotes the following norms for free software development: copyleft, informal management with fluctuating membership, and open communication of all design and code artifacts. The FSM differs from previous computerization movements characterized by Kling and Iacono (1988; 2001) where mass computerization, through investment and deployment of new computing technologies, was the goal of specific computerization movements. Based on the assumption that people are already connected to the Internet, the FSM focuses on proselytizing its ideological frames in an effort to recruit more developers and users of free software with the goal of promoting free software development as the norm, and ultimately eliminating proprietary software.

This vision of the FSM founder, Richard Stallman, is promoted by the FSF, but whether it can succeed in changing the hearts and minds of software developers and users is unclear, and probably unlikely. Why? One, computerization movements are situated within a consumer-based, profit-making society with many software development firms producing proprietary products already embedded in peoples' work and/or home lives. Thus, it would be extremely difficult to convince *all* software users to completely eliminate proprietary software from their computers. However, millions of users have already adopted GNU\Linux based software systems that mix free and open source software, and thus appear comfortable with their choice to avoid or minimize their reliance on proprietary software. Two, it would take a gargantuan undertaking to reproduce all existing proprietary software applications as free software, or alternatively, to convince software development firms to release all source code as free software. Three, would this be an appropriate step to take in situations requiring the highest reliability (e.g. medical equipment that could result in deaths if inappropriately used) or highest security (e.g. systems of national defense)? Actually, there are those who would argue that free or open source software is more reliable than non-free software and that free software is more secure than non-free (e.g. DARPA and the military services are using open source software due to its reputation for being most secure). As time goes on and more and more businesses and government agencies experiment with free and open source software products, the answer to this question regarding software reliability and security will become clearer.

In fact, there will more than likely continue to be a spectrum of believers in the benefits of free software: 1) Some users will embrace free software and the FSM on its own terms; 2) Other users will prefer a mix of free software, open source software, and proprietary with the less restrictive licenses of the OSI; and 3) Some users will not care if the software is free software or open source software as long as it is free, as in free beer.
As free software applications continue to grow in detail and reliability, we predict that more companies will begin investing in free software and, indirectly, to the FSM. In that way, the movement will continue to mobilize free software programmers and users.

What can this research contribute to typical in-house software development projects? For software managers in typical software development environments, this research highlights the strength and importance of community beliefs, values, and norms, as well as organizational culture when directing a large software project. Furthermore, we have shown the value in using the IRC as a means of building community and reinforcing a group's beliefs and norms. Giving programmers an IRC channel for in-house communication regarding software technical issues may foster a sense of camaraderie and organizational culture that might enhance cooperative work and increase productivity. Companies may benefit from taking a small program and creating an in-house open software project to test whether or not a free or open source program would be "better, faster, or cheaper" (Scacchi, 2004). This research is important to software development managers interested in using GNU/Linux or other free software applications in their organization where they may need to deal with an emerging culture (Elliott and Scacchi, 2003b) of free software developers and users.

ACKNOWLEDGMENTS

The research described in this report is supported by grants from the National Science Foundation #0083075, #0205679, #0350754, and #0205724. No endorsement implied. Mark Ackerman at University of Michigan, Les Gasser at University of Illinois, John Noll at Santa Clara University, Chris Jensen and others at the UCI Institute for Software Research are collaborators on this research.

REFERENCES

Ackerman, M. and Halverson, C. (2000), "Reexamining Organizational Memory", *Communications of the ACM,* Vol. 43, No. 1, pp. 59-64.

Blumer, H. (1969), "Social Movements", in McLaughlin, B. (Ed.) *Studies in Social Movements: A Social Psychological Perspective*, Free Press, New York.

Crowston, K. and Scozzi, B. (2002), "Exploring Strengths and Limits on Open Source Software Engineering Processes: A Research Agenda", in *2nd Workshop on Open Source Software Engineering,* Orlando, Florida.

DiBona, C., Ockman, S. and Stone, M. (1999), *Open Sources: Voices from the Open Source Revolution,* O'Reilly & Associates Inc., Sebastol, CA.

Elliott, M. (2000), *Organizational Culture and Computer-Supported Cooperative Work in a Common Information Space: Case Processing in the Criminal Courts,* University of California, Irvine, Irvine, CA.

Elliott, M. (2003), "The Virtual Organizational Culture of a Free Software Development Community", in *3rd Workshop on Open Source Software,* Portland, Oregon.

Elliott, M. and Scacchi, W. (2003a), "Free Software: A Case Study of Software Development in A Virtual Organizational Culture", Institute for Software Research, University of California, Irvine, Irvine, CA.

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

Elliott, M. and Scacchi, W. (2004), "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, Pittsburgh, PA.

Feller, J. and Fitzgerald, B. (2002), *Understanding Open Source Software Development,* Addison-Wesley, New York, NY.

Gregory, K. (1983), "Native-view Paradigms: Multiple Cultures and Culture Conflicts in Organizations", *Administrative Science Quarterly,* Vol. 28, pp. 359-376.

Herbsleb, J. D. and Grinter, R. (1999), "Splitting the Organization and Integrating the Code: Conway's Law Revisited", in *Proc. 21st. Intern. Conf. Software Engineering,* ACM Press, Los Angeles, CA.

Hine, C. (2000), *Virtual Ethnography,* Sage Publications, London.

Iacono, S. and Kling, R. (1996), "Computerization Movements and Tales of Technological Utopianism", in Kling, R. (Ed.) *Computerization and Controversy: Value Conflicts and Social Change, 2nd Ed.*, Academic Press, San Diego, CA.

Iacono, S. and Kling, R. (2001), "Computerization Movements: The Rise of the Internet and Distant Forms of Work", in Yates, J. A. and Maanen, J. V. (Eds.), *Information Technology and Organizational Transformation: History, Rhetoric and Practice*, Sage Publications.

Kling, R. and Iacono, S. (1984), “Computing as an Occasion for Social Control”, *Journal of Social Issues*, 40(3), 77-96.

Kling, R. and Iacono, S. (1988), "The Mobilization of Support for Computerization: The Role of Computerization Movements", *Social Problems,* Vol. 35, No. 3, pp. 226-242.

Kling, R. and Scacchi, W. (1982), "The Web of Computing: Computer Technology as Social Organization", in Yovits, M. (Ed.) *Advances in Computers*, 3-87, Academic Press, New York.

Kollock, P. and Smith, M. (1999), "Communities in Cyberspace", in Smith, M. and Kollock, P. (Eds), *Communities in Cyberspace*, Routledge, London.

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

Martin, J. (2002), *Organizational Culture: Mapping the Terrain,* Sage Publications, Thousand Oaks.

Mockus, A., Fielding, R. T. and Herbsleb, J. (2002), "A Case Study of Open Source Software Development: The Apache Server", *ACM Transactions on Software Engineering and Methodology,* Vol. 11(3), 309-346.

Nardi, B., Whittaker, S. and Bradner, E. (2000), "Interaction and Outeraction: Instant Messaging in Action", in *Conference on Computer Supported Cooperative Work,* ACM Press, Philadelphia, PA.

Norris, J. (2004), “Mission-Critical Development with Open Source Software: Lessons Learned”, *IEEE Software*, January/February 2004, 42-49.

Orlikowski, W. (2000), “Using Technology and Constituting Structures: A Practice Lens for Studying Technology in Organizations”, *Organization Science,* 11(4), 404-428.

Orlikowski, W. and Gash, D. (1994), “Technological Frames: Making Sense of Information Technology in Organizations”, *ACM Transactions on Information Systems*, 12 (2), 174-207.

Orlikowski, W. and Iacono (2001), “Research Commentary: Desperately Seeking the “IT” in IT Research—A Call to Theorizing the IT Artifact”, *Information Systems Research*, 12(2), 121-134.

Pavlicek, R. G. (2000), *Embracing Insanity: Open Source Software Development,* SAMS Publishing, Indianapolis, IN.

Raymond, E. S. (2001), *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O’Reilly, Sebastopol, CA.

Scacchi, W. (2002a), "Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development", Institute for Software Research, UC Irvine, Irvine, CA.

Scacchi, W. (2002b), "Understanding Requirements for Developing Open Source Software Systems", *IEE Proceedings--Software,* Vol. 149, No. 2, pp. 24-39.

Scacchi, W. (2004), "Free and Open Source Development Practices in the Game Community", *IEEE Software,* Vol. 21, No. 1, pp. 59-67.

Schein, E. H. (1992), *Organizational Culture and Leadership,* Jossey-Bass, San Francisco.

Skok W., and Legge, M. (2002), Evaluating Enterprise Resource Planning (ERP) System using an Interpretive Approach, *Knowledge and Process Management*, 9(2), 72-82.

Stallman, R. M. (2002), *Free Software, Free Society: Selected Essays of Richard M. Stallman,* GNU Press.

Strauss, A. and Corbin, J. (1990), *Basics of Qualitative Research: Grounded Theory Procedures and Techniques,* Sage Publications, Newbury Park, California.

Trice, H. M. and Beyer, J. M. (1993), *The Cultures of Work Organizations,* Prentice Hall, Englewood Cliffs, NJ.

Van Maanen, J. V. and Barley, S. R. (1984), "Occupational Communities: Culture and Control in Organizations", *Research in Organizational Behavior,* Vol. 6, 287-365.

Williams, S. (2002), *Free as in Freedom: Richard Stallman's Crusade for Free Software,* O'Reilly & Associates, Sebastopol, CA.

APPENDIX A

Table 2 lists the types of data and the methods used to analyze it. The stages of research refer to the following: 1) Stage 1 – reading and studying of free and open source books, articles, Websites; 2) Stage 2 – site selection, open coding of case studies; 3) Stage 3 – axial coding of three case studies looking for commonalities and differences; comparison of GNUe with other free and open source communities; use of GNUe's search engine to locate discussions regarding ideology in IRC and mailing list archives.

Table 2 – Research Data and Analysis Methods

Type of Data	**Stage/Collection Method**	**Analysis Method**
Books and articles on free and open source software development; Websites of free/open source projects	I – Library and Web searches	Reading and outlining books and articles. Open coding of outlines looking for conceptual framework to help select a research site
Site Data (GNUe and FSF) - IRC - Kernel Cousins - Mailing Lists - Web Site Articles	I – Study of Web site documentation, examined KCs indexed by topic, found topics of conflict and cooperation, selected cases from IRC details. I	Kernel Cousins coded by topics; Cases selected from KCs; IRC and Mailing list archives of 3 cases – open coding; conceptual diagram
Interviews - Email - Face to Face	I (Email) – set up conference F2F meeting II (Face to Face)	Analysis along open codes derived from site data.
Archives of Site Data - IRC - Kernel Cousins - Mailing Lists	III Review of case files again, built graphical conceptual diagram; Review similar KCs and IRCs to test theory.	Axial coding and comparison of GNUe with other free and open source projects. As of early 2004, GNUe upgraded Web site with a Wiki and a sophisticated search engine. Used search to discern similar patterns of work across archives.
Source Code and Documentation; Web site Wiki for GNUe	III Explore more IRC archives to verify theory from earlier research using GNUe search engine; study wiki	Download software for PC; Attempt to install working GNUe unsuccessful to date

The interview protocols were of two types: email and face to face. For the email interviews, two core maintainers who were prominent in the IRC discussions were sent a list of questions via email. One answered quickly and agreed for some follow-up questions:

1. Who decides when a release will occur? Are decisions made by committee or is there one person in charge making top level decisions?
2. Are all contributors non-paid? Are the core maintainers the only ones being paid by consulting firms [part-time]? Do people ever meet face-to-face?
3. In general, what are some of the obstacles and problems that arise in this environment? How are tasks assigned to contributors?
4. Many of the IRC logs show how conflicts arise and are resolved. Is that the main channel for problem solving instead of say, email or phone calls.

Follow-up questions after attending an Open Source conference:

1. Is it common practice to accept criticism from relative newcomers to the work?
2. Are some GNUe programmers motivated by establishing a reputation in the community?
3. How do people use the Kernel Cousins?
4. Can you help us by reviewing our GNUe research papers?

Then about a year into the research, we attended a Linux conference where the same core maintainer was interviewed with the following questions:

1. Open ended at first, interviewee talked for about 10 minutes about free software and GNUe's future.
2. What about IRC? We noticed that you use these to resolve conflicts?
3. What do you do with people who contribute code that is not up to your standards?
4. General questions about decision making about GNUe software development.
5. What do you think of Compiere, the French free ERP software package?
6. What is your motivation to participate in free software? Is building a reputation important?
7. How do you schedule activities in free software projects? Do you know who does what and do you have timelines?
8. Are any other open source projects using IRC as a communication medium?
9. How many companies are using GNUe?
10. How can we help you other than contributing code?

[i] http://www.debian.org
[ii] See http://www.fsf.org/philosophy/free-sw.html for a detailed definition of free software.
[iii] http://www.opensource.org
[iv] www.fsf.org/philosophy/free-software-for-freedom.html
[v] Open Source Definition
[vi] http://www.fsf.org/philosophy/free-sw.html
[vii] For a discussion of licenses recommended by the FSF, see http://www.gnu.org/licenses/licenses.htm. For licenses recommended by the open source initiative, see http://www.opensource.org/licenses/index.php
[viii] http://www.gnuenterprise.org
[ix] http://www.gnuenterprise.org
[x] http://www.gnu.org
[xi] See http://kt.zork.net for the complete set of GNUe kernel cousins.
[xii] http://www.gnue.org
[xiii] http://www.gnu.org/copyleft/gpl.html
[xiv] http://ometer.com/hacking.htm
[xv] http://www.fsf.org/licenses/licenses.html/#WhatISCopyleft
[xvi] http://www.fsf.org/software/devel.htm

When Worlds Collide: Emerging Patterns of Intersection and Segmentation when Computerization Movements Interact

(DRAFT in progress)

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

Abstract

Kling and Iacono introduced and investigated computerization movements, counter computerization movements, and computerization organizations in a series of papers starting in the late 1980's.Their analyses focus on characterizing structural properties, ideologies, and consequences of computerization movements and organizations. This paper starts from their foundations to investigate a set of three new computerization movements: open source software development, computer games, and grid computing. I focus attention on examining not only structural properties, ideologies, and consequences of these three computerization movements and associated organizations, but also on structural processes that characterize what happens when computerization movements come together, and consequences that emerge as a result of these intersecting movements. This focus on identifying emerging patterns of intersecting computerization movements, as well as the segmenting of these computerization movements is the principal contribution of this paper.

Keywords: computerization movements, open source software, computer games, grid computing, patterns of intersection and segmentation.

1. INTRODUCTION

In this paper, I examine three established yet continually emerging worlds of computing. These are the worlds of open source software development [2002], computer games [Scacchi 2004a], and grid computing. Each of these worlds can and has been subject to some of the analytic frameworks that emphasize either ethnographic, empirically grounded theory building or technological rationalization. However, in this paper, I believe it will be most effective to examine what happens as these computerization movements collide or pass through each other in ways that might not be readily predicted from studies of each movement or computing sub-

world onto itself. Subsequently, this study builds on and extends the analytical frameworks of the made by Rob Kling and colleagues over more than 25 years of research in social analyses of computing.

The paper is organized as follows. The next section reviews computerization movements, social movements, and other related framings of collective action within the computing world. It is followed by an examination of three current computerization movements: open source software, computer games, and grid computing. Each movement is examined and compared to prior computerization movements. This reveals trends and structural patterns found in these current movements which differ from those examined in the studies by Kling and Iacono. In particular, attention is focused on how these three current movements intersect each other and the consequences that can follow. This is followed by a discussion that presents some ways for how to further advance studies of computerization movements based on the topics covered in the paper. A statement of conclusions then follow and close with recommendations for new research problems to be investigated related to emerging computerization movements and to the collective actions and social dynamics that animate their movement.

2. UNDERSTANDING COMPUTERIZATION MOVEMENTS

The analyses of Kling and Iacono [1988, 1994, Iacono and Kling 1996, 2001] and others [Elliott and Scacchi 2004] build from a foundation of prior theoretical and empirical studies of social movements. A closer look at the social movements literature reveals additional conceptual foundations that can expand and refine what can be examined when studying social movements, and by extension, when studying computerization movements and counter-computerization movements.

The concept of “social movement” is reasonably well-established within the social sciences. However, like most successful analytical concepts, it is subject to alternative framings and definitions conceived to suit the perspective of its authors. Some of those employed by Kling and Iacono, as well as others can be identified and compared in order to establish a framework for further study of computerization as a social movement.

To start, Kling and Iacono build on the results from Herbert Blumer (1969:8) who indicated that social movements are "collective enterprises to establish a new order of life." Blumer also distinguished between "reform" and "revolutionary" movements. The ideologies of *revolutionary movements* emphasize changing key social relations throughout a social order, while *reform movements* focus on change of a restricted set of social relations. Blumer also distinguished between "specific" and "general" movements. *Specific movements* are segments or sub-movements of a broader, *general movement*. Many social movements like those centered about environmental activism (and its opponents) are heterogeneous. The distinction between specific movements and general movements helps characterize the relationships between distinct segments of a larger movement. But in all cases, Blumer’s studies point to social movements as a form of collective action that may arise inside or outside of existing organizational or institutional forms.

Kling and Iacono also drew on the analysis of Zald and colleagues [Zald and Berger1978, Useem and Zald 1982] to flesh out other properties of social movements that could help characterize computerization movements. For example, Zald and colleagues observed that

social movements can occur within organizations as a participatory strategy for collective action that seeks to affect some organizational reform or major organizational transformation. Such movements may be based on affecting political change within an organization. Alternatively, *organizations may emerge whose purpose is to help guide or lead a movement* in affecting social change through organized collective action. Finally, such an organization may rely on focusing its efforts to mobilizing its movements around certain technologies either in support or opposition to the entrenched technologies of dominant institutions. Zald and colleagues identify these as *technology movements* or *technology counter movements*, depending on whether the organization guiding the social movement supports or opposes the existing technological order. The computerization movements characterized by Kling and Iacono employ these notions of the role of key organizations and technological movements and counter movements.

Many other scholars have studied and theorized about social movements, which can now also influence how we may view computerization movements. For example, Gerlach [1971, 2001] has been engaged in ongoing studies of social movements for more than three decades. He finds that social movements are segmentary, polycentric, and networked, therefore heterogeneous and cyclic. *Segmentary* means that movements are composed of many diverse groups that grow, divide, fuse, proliferate, contract, and die. *Polycentric* means social movements have multiple and sometimes competing leaders or centers of influence, and the persistence of such positions of authority may be short-lived. *Networked* means movements are integrated through multiple relationships among movement participants, the overlapping and joint activities they engage, shared communication media, ideals, and opponents. These networks may therefore be recognized as constituting social networks (including organizations), technological networks (including computing systems and networked information infrastructures), and mutually

dependent socio-technical networks can all therefore exist within technological or computerization movements and counter-movements.

The structural characteristics of segmentary, polycentric, and networked (SPIN), are useful additions to the Kling and Iacono framing that helps flesh out important other structural properties of computerization movements and counter-movements. For example, if computerization movements follow a SPIN cycle, then they must be segmentary rather than monolithic, polycentric rather than singular, and networked rather than hierarchical or fully decentralized and disconnected. This implies that no one set of advocates, groups, or single organization truly speaks for or leads all the participants associated with a computerization movement. Similarly, it implies that conflicts exist within each movement, as do struggles to rest control from those currently in positions of movement leadership.

Kling and Gerson [1977, 1978] introduced the concept of “computing world” as another way to characterize how collective action within the social world of computing is organized and articulated. The computing world effectively combines Blumer's general social movement with Strauss’ [1978] social world perspective and Zald's technological movements, while a given sub-world of computing or occupational community [cf. Gerson 1983, Elliott and Scacchi 2004] whose work centers about distinct kinds of computing systems or technologies, corresponds to Blumer's specific social movement. These papers add an interesting new dimension to that of social movements in that they propose to ground and focus on characterizing movement dynamics within the computing world as being driven in part by *technological innovation practices and processes* [Kling and Gerson 1977, Scacchi 1981]. Broader patterns of technological innovation, resource transactions, computing work trajectories and occupational

careers within and across organizations help articulate patterns of *segmentation and intersection* within the computing world and sub-worlds [Kling and Gerson 1978, Gerson 1983]. The patterns of segmentation and intersection within the computing world and sub-worlds examined by Kling and Gerson reiterate Blumer's and Strauss' foundational notions while corresponding to the SPIN cycle reported by Gerlach. Technological innovation, segmentation, and intersection thus introduce a set of *dynamics that animate and provide a motive force to computerization movements*.

Last, Kling and a loosely coupled group of colleagues (e.g., [Kling, McKim, Fortuna, and King 2000; Kling, McKim, and King 2003; Lamb, Sawyer, and Kling 2000; Rosenbaum 2004, and Scacchi 2004b]) most recently focused their analytical lenses to viewing the emerging patterns of development and deployment of computerization efforts as *socio-technical interaction networks* (STINs). In earlier work, Kling and Scacchi [1979, 1980, 1982] framed their analyses around collective action within and across a "web of computing" to show how the socio-technical worlds of computing segment, intersect, and insinuate themselves through co-evolving organizational regimes, resource dependencies, and technical system arrangements. STINs in turn are juxtaposed as both an evolutionary extension of the web of computing framework, and as an alternative to the Actor-Network Theory (ANT) framework developed by Latour [1987], Callon, Law [1996] and others, found in many recent social studies of science and technology.

Collectively, these studies reveal a sustained intellectual focus of computing, computing world, and computerization movements as a complex socio-technical regime of organizational and technological resources arrayed through recurring patterns of negotiation, reallocation, and control that simultaneously enable and constrain what people can accomplish in their work with

computing. But computing and computerization continues to evolve and new regimes and movements are emerging, which may or may not change the landscape of analytical framings that have appeared, and thus merit study and reapplication so as to re-establish their relevancy and theoretical validity.

3. Three Emerging Computerization Movements

The computerization movement studies of Kling and Iacono examine initially five movements; urban information systems, artificial intelligence, personal computing, office automation, and computer-based education [Kling and Iacono 1988]. They later added two additional computerization movements, for virtual reality and computer supported cooperative work (CSCW). For the most part, all of these computerization movements, with the exception of CSCW have mostly receded in their prominence or aggressive promotion. Why and how these computerization movements have faded is an open question beyond the scope of this paper. However, it is fair to say that it may be explained in part by reference to the comparative success or failure of these early computerization movements to facilitate the broad diffusion of computing innovations that are often the focus of each respective computerization movement [cf. Rogers 1995]. However, none of the studies by Kling and Iacono examined the dynamics that drove these movements, nor whether any of these computerization movements drove into one another and to what consequence. This is the point of departure in the remainder of the study presented here.

Specifically, we can examine a new set of computerization movements, not only in terms of their structural properties and ideological foci as outlined in the previous section, but also to some of their dynamics, including attention to where and how different computerization

movements may intersect each other, and to what ends. Accordingly, three computerization movements now in the limelight can be examined and compared, both as separate movements, and as intersecting or overlapping movements. The three computerization movements for study are those focused on open source software development, networked computer games, and grid computing.

3.1 Open Source Software

3.1.1 Structural patterns of the OSS movement

The OSS movement is populated with thousands of OSS development projects, each with its own Web site. Whether the OSS movement is better recognized as a countermovement 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]. Nonetheless, more than 100K such projects are registered at OSS portals like SourceForge.org, Freshment.org, and Savannah.org, as seen in Exhibit 1. The vast majority of these projects appear to be inactive, with less than 2 contributing developers. A few thousand OSS projects seem to garner most of the attention and community participation, but no one project defines or leads the movement. The Linux Kernel project is perhaps the most widely known, with its celebrity leaders, like Linus Torvalds. 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 empirical studies [cf. Koch 2005] 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 of the OSS movement, but at the same time, the multiplicity of projects reflects is segmentation.

3.1.2 Ideological tenets of the OSS movement

The OSS movement arose in the 1990's [DiBona, *et al*., 1999] from the smaller, more fervent "free software" movement [Gay 2003] started in the mid 1980's. The free software movement [Elliott and Scacchi 2004] was initiated by Richard M. Stallman [Gay 2003], and its members 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 "copyleft"), 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 60% or more of all OSS projects (including the Linux Kernel project) employ the GPL, even though there are only a few thousand of self-declared free software projects. OSS projects, like 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. In simple terms, free software is always available as open source, but open source software is not always free software. So the free software movement has emerged or has been subsumed as a sub-world within the larger 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. It also distinguishes those who identify themselves as OSS developers, but not practitioners or affiliates of the free software sub-world.

3.1.3 Organizations of the OSS 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 2003]. The OSI's Bruce Perens who advocates that OSS is an viable economic and innovative alternative to proprietary software, often is juxtaposed or compared to the FSF's Richard M. Stallman, who seeks to "put back the *free* in free enterprise" [Gay 2003]. 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].

3.2 Computer games

3.2.1 Structural patterns of the Computer Games movement

Computer games[1] have become a pervasive element of popular culture. More than 100 million computers have been sold for computer gaming applications, and every PC, PDA, and cell phone now comes with computer games installed [King 2002]. More than 500M users have played computer games over the Internet at one time, and upwards of 2.5M users play networked multi-player games per day, as indicated in Exhibit 2. Furthermore, the currently most popular networked games like *Half-Life: Counter-Strike* and *World of Warcraft* have millions of online players, while the most popular single player games (or game-based synthetic worlds) like *The Sims* have tens of millions of players[2]. Computer games are, safe to say, a global entertainment technology, and one that increasingly defines the leading edge of personal computing technology.[3] However, computer game technology in general, and computer games in particular has little presence or advocacy within academia, or within computer science

[1] Historically, many authors identify computer games as "video games" as if to suggest their significance stems from their video/visual display, rather than from their basis as a computer or computing system.
[2] It is also worth noting that parlor games like Solitaire, Minesweeper, and Pinball may be even more pervasively deployed, being found on hundreds of millions of personal computers and related devices.
[3] This can be seen in that the most expensive PCs are those configured for high-performance computer gaming.

research laboratories. This stands in contrast to other computerization movements like Artificial Intelligence or Personal Computing, when they first appeared, which clearly had a base of support within the academic and research communities. Similarly, it is somewhat challenging to find whether avid computer game players or game developers identify themselves as part of either a revolutionary or reform movement, rather than people who enjoy using computers to play games and have fun. If anything, this makes it seem that if there is a computer game movement, it might be one that seeks to reform the vision of computers as simply instrumental devices that support administrative or technical work tasks, to one where computers can be treated as hedonistic devices that support fun, playful competition, and sustained periods of immersive fantasy and use. The Computer Games movement might then appear to be one focus on diffused cultural and technological change, rather than sharply targeting organizational or institutional change. But a closer look may reveal more than this.

3.2.2 Ideological tenets of the Computer Games movement

As elements of popular culture, computer games would not seem to constitute a social movement, any more than some other personal technology (transistor radios, pocket calculators, or portable music players). Further, the popular press associated with computer games, periodicals like *PC Gamer*, *Electronic Gaming World*, and dozens of other similar titles, or even trade titles like *Game Developer* do not seem to convey that computer games are in some way a social force or mechanism for social change; instead computer games are about fun and play, and something entertaining to do for those who have grown up in the modern computer age. Furthermore, the technology, design, and development practices of commercial game developers seem more strongly aligned (or gender biased) to the interests of young male adults [Cassell and Jenkins 1999], and such interests would not typically be recognized as a basis for a

social movement. However, there is growing interest from humanists and business pundits who are beginning to articulate a vision that computer games have the potential to be part of a dramatic transformation of culture, entertainment, and online social interaction. For example, Prensky [2001], as well as Beck and Wade [2004], see computer games as a precursor for a new generation of business computing applications that will transform how modern businesses operate. Prensky [2001] and Gee [2003] see similar transformations arising within the institution of education, in that computer games also appear to operate as immersive learning and persistent socialization environments. Gee [2003], and also Wolf and Perron [2004], see computer games emerging as a new cultural medium, much like radio, television, and cinema, which may transform the language, experience, and venues for communication, learning and social action. Such game-based computing environments or media may therefore represent a new force giving rise to educational reform, as well as to new ways and means for learning on the job or on demand. So it appears that the ideological basis for a social movement centered about computer games is emerging from outside of its industrial centers of development, and from academic interests not typically associated with promoting information technology.

3.2.3 Organizations of the Computer Games movement

The computer game industry already generates larger annual revenues (about $10B in 2004) than the feature film industry (less than $9B in 2004), and computer games (software) are often the leading edge embodiments of game consoles and personal gaming computers (hardware). Large corporations like Electronic Arts, Microsoft, Sony, and Nintendo indicate the international basis of this technology, and countries like Korea probably have the highest per capita penetration of computer games with broadband network connectivity. Small game design studios like Id Software, Epic Games, BioWare, Valve Software, and Sony Online

Entertainment in the U.S., and NC Soft in Korea are responsible for much of the software technology and "game engines" that are the core of computer games. In contrast, popular game titles marketed in retail venues like Best Buy, Circuit City, Amazon.com and WalMart in the U.S., are distributed by large firms like Electronic Arts, Ubisoft, Atari, THQ, Microsoft, and others that effectively control the retail channels for computer games, and thus what games become widely available on a regional, national, or international basis. Furthermore, other large IT companies like IBM, SUN, Dell, and HP are all now shipping both personal computers and clustered server systems that are preconfigured and packaged to support computer games or game-centered businesses (e.g., online networked game play service providers) for sale to individual or corporate buyers.

Mid-size semiconductor manufacturing firms like NVidia and ATI now dominate the market for devices specialized to support computer game graphics (graphics processing units, GPUs), while these GPUs now represent the most complex digital processors in the computer industry. Small, boutique personal computer vendors like Alienware, Voodoo, Falcon, and others have emerged and thrive on selling highly custom personal computers that are configured and optimized (including "overclocked" CPUs and GPUs). Elsewhere, as a small but growing number of massively multiplayer online games (MMOG) and online role-playing games (MMORPG) like *Everquest, Ultima Outline, World of Warcraft (WoW),* and *Second Life*. In the case of the first three of these MMORPG, their parent companies have realized many millions of dollars of revenues not only from sale of the games, but also from pay-to-play online subscriptions from players who subscribe on a monthly basis. In the case of *WoW*, the most popular subscription-based MMORPG in the U.S. and other parts of the world, more than 4M players pay between $11-15/month to play the game and participate in the online experience

and community of *WoW*. Furthermore, these MMORPG have emerged on the international scene offering not only persistent (24/7) online game play worlds, but also real-world economic systems that are generating income and wealth comparable in gross domestic products terms to developed nations (the virtual game world of *Norrath* associated with the MMORPG *Everquest*, now boasts of a GDP monetized in U.S. dollars at approximately $2,600/year, placing its economy ahead of the nation of China, and just behind that of Russia [cf. Castronova 2005]).[4]

Next, the U.S. Department of Defense is now investing heavily into the development and distribution of computer games as interactive media and educational technologies for conveying the modern military (combat) experience, as well as for training troops in small team tactical warfare. The game, *America's Army*, which is available for free download from the Web, has become the most widely distributed networked computer game in history, with more than 20 million copies in circulation, and nearly 6 million users registered on its associated Web portal, who play on one or more of the 40K AA servers accessible over the Internet. However, it is difficult to find examples in other government agencies involved in funding research and development (e.g., NSF, NASA, DoE, NIH) that are investing in computer games or game-based applications.

Finally, there are a yet unaccounted number of Web portals or Web sites supported by game players. These are generally "virtual organizations" [DeSanctis and Monge 1999, Tuecke, Foster, Kesselman 2001] whose primary organizational form is based on the use of electronic

[4] The market for producing and selling in-game resources, assets, or high-rank game characters can be seen, for example, in postings found on EBay.com, where top-price items for games like Everquest or WoW range from hundreds to thousands of U.S. dollars, even though the end-user license agreements associated with these games may restrict or prohibit such unauthorized trade of copyrighted game properties.

communication systems and media like Email, Web sites, discussion forums, Weblogs (blogs), and others. However, the game related virtual organizations are generally not going concerns organized for financial gain or capital growth, but instead are organized as online venues for social interaction and community interaction around favorite games. Game development companies appreciate the economic and market development value of these online communities, and sometimes actively sponsor or host them on their corporate Web servers [cf. Kim 2000]. Three types of virtual organizations for games include fan sites, clan sites, and tournament sites. Fan sites attract and organize the efforts of game players who want to share their game play experiences, results, or creations (e.g., game-related artworks) with other like-minded enthusiasts. Clan sites draw the attention of avid or hard-core game players who want to be identified with a group of like-minded, accomplished players or game modders [Cleveland 2001] in order to advance their game play or game development skills. Player-based clans seek to be able to subsequently engage in team-oriented (clan versus clan) play within multi-player games. Such multi-team engagements, as well as advanced player versus player engagements, are facilitated through tournament sites, which seek to elicit top-tier game players in professional or near-professional levels of game play. The most popular networked, multi-player games often have hundreds of fan and clan sites that identify themselves with a particular game, or set of related games, while tournament sites are often associated with either a specific game or game vendor (QuakeCon.org for games from Id Software, BlizzCon.com for games from Blizzard Entertainment), or for regional, national, or international game play events,[5] and thus also bring along corporate sponsors to finance the event, and to showcase their products to game players.

[5] The *World Cyber Games* (based in Korea) is an example of an international league of national game teams made

3.3 Grid Computing

3.3.1 Structural patterns of the Grid Computing movement

Compared to the other two computerization movements discussed here, grid computing is a computerization movement whose activities and promoters are much more like the computerization movements examined in Kling and Iacono's studies. The advocates of Grid Computing envision a new order for how large-scale enterprise computing should be structured in terms of hardware, software, and networking. Grid computing is envisioned to be based on loosely-coupled/distributed computer clusters and shared storage systems (hardware), grid-based middleware services and remote application services (software), and high throughput data networking [Foster 2002, Johnson, *et al.,* 2004]. An enterprise that plans to adopt grid computing is one that must plan to make a major investment in new computing technologies and development services. Ironically, the major reason most commonly cited for such investment is cost reduction and resource flexibility, which are to be realized through migration from monolithic legacy system applications, to loosely-coupled and incrementally reconfigured applications that are composed from "best of breed" applications services. So grid computing is supposed to realize its benefits through offering an enterprise a new, agile computing application environment, one that might be adapted to meet the ebb and flow of business cycles [Johnson, *et al*., 2004]. In this way, application services can scale up to meet growth demands, or can scale down to shrinking demands. However, all of these capabilities are mostly yet to be demonstrated in real-world business settings where agility or adaptability is critical to an enterprise's operations or success.

---

up of game play enthusiasts who play for cash prizes (up to $50,000 winner per game, per event), gaming equipment, and product endorsements.

3.3.2 Ideological tenets of the Grid Computing movement

Perhaps the most common message associated with Grid Computing is that it represents the future of enterprise computing [Foster and Kesselman 2003]. Grid Computing is not in general advocated as personal or small group technology; instead, it is intended primarily for large enterprises with large IT budgets or investments. For many years, computing grids remained the playthings of researchers, but now in 2005, they are said to have finally come of age [Worthington 2005]. The key to grid computing is said to be focused on "federating" existing computing resources, thereby breaking down the technological boundaries that make enterprise computing more expensive and less reliable than it should be [Foster and Kesselman 2003, Worthington 2005]. Similarly, Grid Computing is about enabling the configuration and reconfiguration of virtual organizations [DeSanctis and Monge 1999, Tuecke, Foster, Kesselman 2001], whereby it becomes possible to configure computing grids that transcend the boundaries of the (physical) organizations where they reside. However, such a capability seems to ignore the long legacy of research into organizations and organization science, let alone the politics of organizations, organizational computing, resource fiefdoms, or even science data wars [Hunter 2003].

3.3.3 Organizations of the Grid Computing movement

Compared to the other two computerization movements discussed here, the grid computing movement appears to be smaller in terms of the number of participating organizations, but those that do tend to be primarily large enterprises, consortia, or research laboratories. Grid Computing is an emerging commercial marketplace in the IT industry, so all the major IT firms

like IBM, SUN, Microsoft, HP, Oracle, and others are creating hardware-software-networking product lines that embrace Grid Computing (or "Web services") technologies. The GRID Forum (www.gridforum.org) is a trade association interested in promoting grid technologies for scientific and commercial computing applications, as well as hosting conferences and trade shows to help promote the commercialization and standardization of these technologies. Finally, most government agencies involved in funding research and development in the U.S., Europe, and beyond, are all investing in R&D projects that seek to stimulate the deployment, growth, and standardization of grid computing technologies, especially in support of large science laboratories or major science research projects (e.g., in areas like genomics, high energy physics, astrophysics, computational chemistry, and others). However, the large IT consultancies like Accenture, EDS, PriceWaterhouse Coopers, and others are yet to provide large-scale service offerings based on grid computing platforms or technologies, compared to their (highly profitable) service offerings based on software technologies like enterprise resource planning (ERP), or customer relationship management (CRM) systems.

4. INTERSECTING COMPUTERIZATION MOVEMENTS

Just as people at work participate in multiple social worlds, they might also participate in multiple computing sub-worlds, and thus in multiple computerization movements, as these computing worlds collide [cf. Kazmer and Haythornwaite 2001]. This section explores and characterizes the structural patterns and dynamic processes that arise when otherwise independent computerization movements intersect one another, as well as the segments that may form and persist as a result of the intersection.

4.1 OSS and Computer Games

The world of OSS and computer games is an active area of engagement and collective action [Scacchi 2004a]. These two independent computerization movements clearly intersect each other. Similarly, there are well established and easy to identify segments within these intersecting movements in the form of OSS-based Computer Games projects found on the Web. For example, on the SourceForge.net Web portal, there are over 10,000 self-declared OSS-based Computer Games projects, out of the 100K OSS projects, as seen in Exhibit 3. However, this 10% sub-world is only the fifth largest community of interest within SourceForge. Nonetheless, the existence of on the order of 10,000 OSS computer game development projects points to the emergence of a computing world that apparently conflates the fun and play of computer games, with the community participation and technical development work of OSS development. Said differently, if working as a software developer for personal enrichment, then developing and playing computer games should be fun. It might also be a provocative and non-traditional way to learn about computer science, software development, and computer game design [cf. Prensky 2001, Gee 2003], well outside of established educational institutions. But whether it leads to a productive professional career as a commercial software developer or business venture is unclear. However, it can in fact provide an entry into the computer game industry, as OSS-based computer game development, through demonstration of "game modding" [Cleveland 2001] or innovative game design experience [Scacchi 2004a, Game Developer 2004].

This intersecting movement is highly active compared to those that follow. It appears that this sub-world could emerge into its own separate computerization movement, with its own innovation practices, ideological beliefs, and internal segmentation and intersection with other

independent computerization movements. These are described in a related study [Scacchi 2004a].

4.2 OSS and Grid Computing

OSS and Grid Computing, most often referred to as "open grid services", are intersecting computerization movements at a scale smaller that OSS and Computer Games, but larger than Computer Games and Grid Computing. It appears that this sub-world will not emerge into a distinct computerization movement, but instead will more likely be assimilated back into the world of Grid Computing. Reasons for why this may occur include the following. First, the software core of Grid Computing is the middleware technology called *Globus* [Foster and Kesselman 2003]. Globus has been and still remains as OSS, as indicated in Exhibit 4. The choice to open source Globus was tied to the desire to have an open standard for defining and integrating grid services (aka, Web services), as well as integrating with both new and pre-existing open data communication protocols. Similarly, the nascent effort to establish ad hoc "flash mob computing" services (cf. www.flashmobcomputing.org), which entail the rapid assembly of supercomputing clusters from networked and participant-contributed PCs, seems to have failed to emerge as a sustainable grid-like computing infrastructure, even though many OSS technologies have been marshaled and configured to demonstrate the potential.

4.3 Computer Games and Grid Computing

Computer Games and Grid Computing are intersecting computerization movements at the scale smaller than the preceding two. But traces of their intersection and emerging persistent segments that span and relate the two can be found and examined [Levine and Wirt 2003]. Similarly, the Sony Corporation has announced that its new, yet-to-be-released PlayStation3

computer game console[6], is designed to be hardware, software, and network compatible with grid computing [cf. Worthington 2005]. Study of this segment reveals a narrow set of recurring practices and innovation processes that appear to limit the near-term growth into a separate computerization movement. It appears that this sub-world may emerge into a distinct computerization movement that will more likely be assimilated back into the world of Computer Games. Reasons for why this may occur arise from recognition that the market for grid computing remains uncertain outside of scientific research applications, while the revenues generated by the computer game industry already exceed $10B/year. Furthermore, the emergence of the MMOG and MMORPG-based persistent online worlds and external economic systems point to plausible opportunities for marketing and deploying computer game-oriented grid systems to national or international game service providers. At the same time, computer game grid represent a new arena for technical innovation by game developers, as well as an arena for new kinds of computer game play experiences for end-users that may conflate having fun with making money (or not!). Finally, it may represent a bundling of technologies that governmental agencies that fund advance educational research (e.g., NSF) may find to be an applying venue for exploring new concepts in large-scale game-based learning environments. Such applications may find interest more within the computer science research community, rather than in education or humanities programs which are helping to motivate the opportunity.

4.4 OSS, Computer Games, and Grid Computing

This is the smallest and least populated sub-world of computing that spans each of the three computerization movements. The fact that there are any identifiable and persisting activities within this segment is noteworthy and significant, since they must interlink three otherwise

[6] Sony has already sold more than 100,000,000 PlayStation1 and PlayStation2 computer game consoles

independent computerization movements. There are no established projects or corporate ventures in this arena, though the idea of their existence seems to be in circulation. Whether this community can emerge, survive, and achieve the critical mass of social networking that can precipitate reinforcing network externalities is an open question. However, examining how participants in this segment can or might interact collectively, and whether there is a common overlapping ideology is of interest. For example, there are many large-scale "LAN parties" where computer game players congregate (i.e., move to a common location) to build truly large-scale local area networks and interconnected game servers to play both commercial games and OSS game mods, and then disappear all in a matter of days, much like when the circus comes into a town. Exhibit 5 portrays such an example from QuakeCon, one of the largest annual gaming enclaves and social event, formed about a large-scale LAN-based computing environment similar in capability to a flash mob computing cluster. Subsequently, this nascent sub-world of computerization that intersects OSS, computer games, and grid computing is perhaps most interesting as a boundary case whose future or continuing emergence is unclear, as is it status as a viable computerization movement. Said differently, it may represent a case where the conditions needed to support the emergence of a computerization movement are partially articulated, but whose longer-term success or demise is unclear. Thus sub-world this represents a case for further study.

5. DISCUSSION and CONCLUSIONS

In this paper, I sought to examine computerization movements surrounding open source software, computer games, and grid computing, none of which had yet been subjected to critical analysis. I also adopted an updated perspective on computerization movements in terms that

---

worldwide.

seek to reveal the segmented, polycentric, and networked dimensions of the social worlds and sub-worlds that constitute these movements. In doing so, it became possible not only to examine each of these computerization movements on their own, but also to explore whether and how these movements might intersect one another, and thus manifest inter-movement conditions and dynamics. This kind of analysis was suggested by early works of Kling and Gerson [1977, 1978] more than 25 years ago, but this analysis strategy still proves viable, as well as a source of new insights into how computerization movements are animated and evolve. Perhaps this is due to the growing maturity and ubiquity of computing technology and culture, as well as to our ability to critically examine them more closely.

This analysis also helps reveal that computerization movements and computing worlds are more diverse and more complexly structured than perhaps may have been seen in prior studies. Also, the outcome of the intersection of different computerization movements is not uniform, and does not necessarily give rise to a new, persistent sub-world of computing, though new sub-worlds do arise, as appears to be the case of the world of open source software for computer games. In fact, just the opposite can happen—the inertia of one computing world effectively subsumes the contribution of the intersecting segment, resulting in the two former sub-worlds remaining co-existent as mostly distinct movements. This appears to be the situation resulting from the intersection of the sub-world of OSS and grid computing, as well as grid computing and computer games. Finally, the nascent intersection of these three computerization movements points to a fledging sub-world whose sustained existence is sufficiently unclear and ill-defined to determine whether any of the three computerization movements will likely assimilate it. Subsequently, this homeless sub-world perhaps becomes most interesting from a critical perspective because of how it helps reveal and deconstruct the boundary conditions that arise as computing worlds and marginal social movements intersect one another.

Established groups within a computerization movement can and often will fragment, resulting in either new groups with different members and allies, or in the departure of the disenfranchised. Consequently, computerization movements are probably more heterogeneous rather than homogeneous, though collective action still emerges, and shared beliefs or ideologies are expressed and renewed through communication media shared within a given movement. Thus, computerization movements must be examined in ways that highlight their heterogeneity, their segmented social worlds, and the socio-technical interaction networks that enable these segments to collective act toward partially articulated and often conflicting goals. This revised perspective seeks to avoid or mitigate assumptions that all participants who are identified or associated with some computerization movement or countermovement myopically subscribe to some master narrative of ideological beliefs or uncritically agree to follow prescribed courses of collective action without reflection, disagreement, consternation, or conflict.

6. ACKNOWLEDGEMENTS

The research described in this report is supported by grants #0083075, #0205679, #0205724, and #0350754 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; and Margaret Elliott, Chris Jensen and others at the UCI Institute for Software Research are collaborators on this research effort.

7. REFERENCES

Beck, J.C. and Wade, M., (2004), *Got Game: How the Gamer Generation is Reshaping Business Forever*, Harvard Business School Press, Cambridge, MA.

Blumer, H. (1969). Social Movements. In B. McLaughlin (Ed.), *Studies in Social Movements: a Social Psychological Perspective*. (pp. 8-29). New York: Free Press.

Callon, M., Law, J., and Rip, J., (eds.), (1996). *Mapping the Dynamics of Science and Technology: Sociology of Science in the Real World.* London: Macmillan Press.

Cassell, J. and Jenkins, H., (eds.), (1999), *From Barbie to Mortal Kombat: Gender and Computer Games,* MIT Press, Cambridge, MA.

Castronova, E. (2005). *Synthetic Worlds, The Business and Culture of Online Games,* University of Chicago Press, Chicago, IL.

Cleveland, C. (2001). The Past, Present, and Future of PC Mod Development, *Game Developer*, 8(2), 46-49, February.

DeSanctis, G. and Monge, P. (1999), Introduction to the Special Issue: Communication Processes in Virtual Organizations, *Organization Science*, 10(6), 693-703.

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

Elliott, M.S. and Scacchi, W. (2004). Mobilization of Software Developers: The Free Software Movement, revised version to appear in *Information, Technology and People*.

Fink, M., (2003), *The Business and Economics of Linux and Open Source*, Prentice Hall PTR, Upper Saddle, NJ.

Foster, I., (2003). What is the Grid? A Three Point Checklist, White paper, Univa Inc.

Foster, I. And Kesselman, C., (eds.), (2003), *The GRID 2: Blueprint for a New Computing Infrastructure,* Morgan Kaufman, San Francisco, CA.

Game Developer, (2004), http://www.gdmag.com/homepage.htm

Gay, J. (Ed.), (2002), *Free Software Free Society: Selected Essays of Richard M. Stallman*, GNU Press, Free Software Foundation, Boston, MA.

Gee, J., (2003), *What Videogames have to Teach Us about Learning and Literacy*, Palgrave Macmillan, New York.

Gerlach, L.P. (1971). Movements of Revolutionary Change: Some Structural Characteristics, *American Behavior Scientist*, 14, 812-836.

Gerlach, L.P. (2001). The Structure of Social Movements: Environmental Activism and Its Opponents, in Arquilla, J. and Ronfelt, D., (eds.), *Networks and Netwars: The Future of Terror, Crime, and Militancy*, The Rand Corporation, Santa Monica, CA.

Gerson, E.M. (1983). Scientific Work and Social Worlds, *Knowledge: Creation, Diffusion and Utilization*, 4(3), 357-377.

Gomes, L. (2001). In your face! MS open source attack backfire, *The Wall Street Journal Online*, 13 June 2001.

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.

Hunter, P. (2003). Datawars: Grid Computing Democratizes Proteomics; Security Concerns Limit Some Efforts, *The Scientist*, 17(8), 53-54.

Iacono, C.S. and Kling, R., (1996). Computerization Movements and Tales of Technological Utopianism, in Kling, R., (Ed.), *Value Conflicts and Social Change*, 2nd. Edition, Academic Press, San Diego, CA.

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.

Joseph, J., Ernest, M. and Fellenstein, C. (2004). Evolution of Grid Computing Architecture and Grid Adoption Models, *IBM Systems J.* 43(4), 624-645.

Kazmer, M.M. and Haythornwaite, C. (2001). Juggling Multiple Social Worlds, *American Behavior Scientist*, 45(3), 510-529.

Kim, A.J. (2000), *Community Building on the Web : Secret Strategies for Successful Online Communities*, Peachpit Press, Berkeley, CA**.**

King, L. (Ed.), (2002), *Game On: The History and Culture of Videogames*, Universe, New York.

Kling, R. and Gerson, E.M., (1977). The Social Dynamics of Technical Innovation in the Computing World, *Symbolic Interaction*, 1(1), 132-146.

Kling, R. and Gerson, E.M., (1978). Patterns of Segmentation and Intersection in the Computing World, *Symbolic Interaction*, 1(2), 24-43.

Kling, R., and Iacono, C.S., (1988). The Mobilization of Support for Computerization: The Role of Computerization Movements, *Social Problems*, 35(3), 226-242.

Kling, R., and Iacono, C.S., (1994). Computerization Movements and the Mobilization of Support for Computerization, in S.L. Star, (ed.), *Ecologies of Knowledge*. SUNY Press.

Kling, R. and Scacchi, (1979). W., Recurrent Dilemmas of Computer Use in Complex Organizations, *Proc. National Computer Conf.*, 48, 107-116, AFIPS Press, New York, NY.

Kling, R. and Scacchi, W., (1980). Computing as Social Action: The Social Dynamics of Computing in Complex Organizations, in Yovits, M. (ed.), *Advances in Computers*, 19, Academic Press, NY.

Kling, R. and Scacchi, W., (1982). The Web of Computing: Computer Technology as Social Organization, in Yovits, M. (ed.), *Advances in Computers*, 21, 1-90, Academic Press, NY.

Kling, R., McKim, G., and King, A. (2003). A Bit More to IT: Scientific Multiple Media Communication Forums as Socio-Technical Interaction Networks**,** *J. American Society Information Science, 54(1)*, 47-67.

Koch, S. (Ed.), (2004), *Free/Open Source Software Development*, Idea Group Publishing, Hershey, PA.

Lamb, R., Sawyer, S., and Kling, R., (2000). A Social Informatics Perspective On Socio-Technical Networks, in H. Michael Chung, (Ed.), *Proc. Americas Conf. Information Systems*, Long Beach, CA.

Latour, B., (1987). *Science in Action*, Cambridge, MA, Harvard University Press.

Levine, D. and Wirt, M. (2003) Interactivity with Scalability: Infrastructure for Multiplayer Games, in Foster, I. and Kesselman, C. (2003).

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

Prensky, M., (2001). *Digital Game-based Learning*, McGraw-Hill, New York.

Rogers, E., (1995). *The Diffusion of Innovations (4th Edition)*, Free Press, New York.

Rosenbaum, H. (2004), Socio-Technical Interaction Networks as a Tool for Understanding Digital Libraries, Paper presented at *ASIST 2004 Annual Meeting*; "Managing and Enhancing Information: Cultures and Conflicts" (ASIST AM 04), Providence, RI.

Scacchi, W. (1981). *The Process of Innovation in Computing: A Study in the Social Dynamics of Computing*, unpublished Ph.D. dissertation, Information and Computer Science Department, University of California, Irvine, Irvine, CA.

Scacchi, W. (2002). Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39, February.

Scacchi, W., (2004a). Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February.

Scacchi, W., (2004b). Socio-Technical Interaction Networks in Free/Open Source Software

Development Processes, in S.T. Acuña and N. Juristo (Eds.), *Peopleware and the Software Process*, World Scientific Press, to appear.

Strauss, A., (1978). A Social World Perspective, *Studies in Symbolic Interaction*, 1, 119-128.

Tuecke, S., Foster, I., and Kesselman, C., (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *Intern. J. Supercomputing Applications*, 15(3), 2001.

Useem, B. and Zald, M. (1982). From Pressure Group to Social Movement: Organizational Dilemmas of the Effort to Promote Nuclear Power. *Social Problems*, 30(2), 144-156.

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.

Wolf, M.J.P. and Perron, B., (2003), *The Video Game Theory Reader*, Routledge, New York.

Worthington, D. (2005), Interview: The Future of Grid Computing, *Beta News*, 21 February 2005.

Zald, M. and Berger, M. (1978). Social Movements in Organizations: Coup d'Etat, Insurgency, and Mass Movements, *American Journal of Sociology* 83(4), 823-861.

EXHIBITS:

Exhibit 1. Home page of the SourceForge.net OSS Web portal, indicating nearly 100K registered projects (source: http://sourceforge.net/, February 2005).

Exhibit 2. Some descriptive statistics characterizing Internet-based use of multi-player computer games (Source: http://www/csports.net, October 2005).

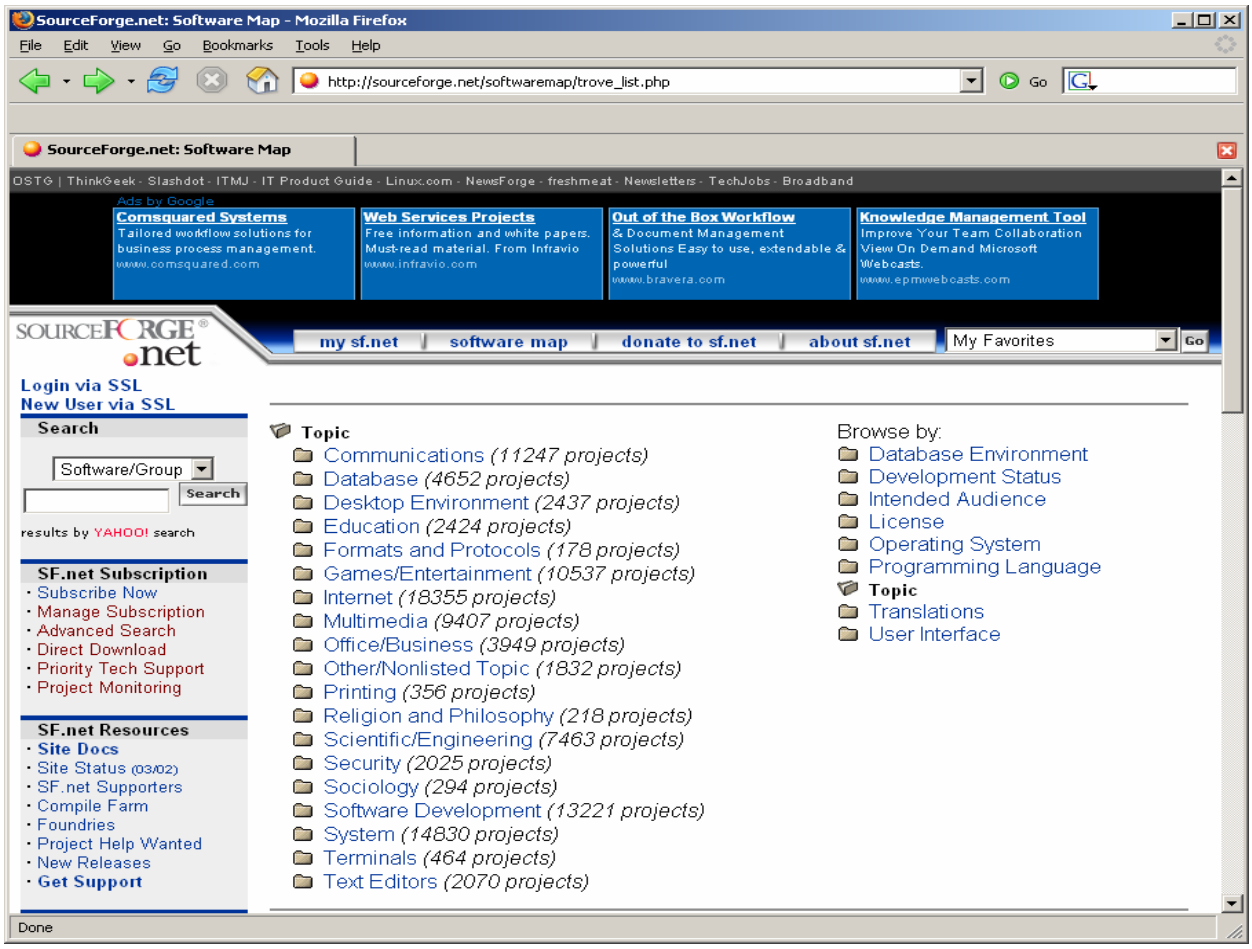


Exhibit 3. Screen display from SourceForge.net indicating the "Games/Entertainment" OSS project category includes more than 10,000 projects (source: http://sourceforge.net/softwaremap/trove_list.php, February 2005).

Exhibit 4. Background information on the OSS Globus Toolkit for building Grid Computing applications (source: http://www-unix.globus.org/toolkit , February 2005.)

Exhibit 5. Overview of a large, informal computing grid for playing commercial computer games and open source game mods found at the QuakeCon2002 LAN Party. (Source: http://gallery.shrocks.com/quakecon2002?page=7)